4 objdump

```
objdump [-a|--archive-headers]
        [-b bfdname|--target=bfdname]
        [-C|--demangle[=style] ]
        [-d|--disassemble[=symbol]]
        [-D|--disassemble-all]
        [-z|--disassemble-zeroes]
        [-EB|-EL|--endian={big | little }]
        [-f|--file-headers]
        [-F|--file-offsets]
        [--file-start-context]
        [-g|--debugging]
        [-e|--debugging-tags]
        [-h|--section-headers|--headers]
        [-i|--info]
        [-j section|--section=section]
        [-l|--line-numbers]
        [-S|--source]
        [--source-comment[=text]]
        [-m machine|--architecture=machine]
        [-M options|--disassembler-options=options]
        [-p|--private-headers]
        [-P options|--private=options]
        [-r|--reloc]
        [-R|--dynamic-reloc]
        [-s|--full-contents]
        [-W[lLiaprmfFsoRtUuTgAckK]|
         --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges,=pubtypes,=trace_info,=trace_abbrev,=trace_aranges,=gdb_index,=addr,=cu_index,=links,=follow-links]]
        [--ctf=section]
        [-G|--stabs]
        [-t|--syms]
        [-T|--dynamic-syms]
        [-x|--all-headers]
        [-w|--wide]
        [--start-address=address]
        [--stop-address=address]
        [--prefix-addresses]
        [--[no-]show-raw-insn]
        [--adjust-vma=offset]
        [--dwarf-depth=n]
        [--dwarf-start=n]
        [--ctf-parent=section]
        [--ctf-symbols=section]
        [--ctf-strings=section]
        [--no-recurse-limit|--recurse-limit]
        [--special-syms]
        [--prefix=prefix]
        [--prefix-strip=level]
        [--insn-width=width]
        [-V|--version]
        [-H|--help]
        objfile…
```

`objdump` displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

objfile… are the object files to be examined. When you specify archives, `objdump` shows information on each of the member object files.

The long and short forms of options, shown here as alternatives, are equivalent. At least one option from the list `-a,-d,-D,-e,-f,-g,-G,-h,-H,-p,-P,-r,-R,-s,-S,-t,-T,-V,-x` must be given.

`-a`
`--archive-header`

If any of the *objfile* files are archives, display the archive header information (in a format similar to '`ls -l`'). Besides the information you could list with '`ar tv`', '`objdump -a`' shows the object file format of each archive member.

`--adjust-vma=`*offset*

When dumping information, first add *offset* to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

`-b` *bfdname*
`--target=`*bfdname*

Specify that the object-code format for the object files is *bfdname*. This option may not be necessary; *objdump* can automatically recognize many formats.

For example,

```
objdump -b oasys -m vax -h fu.o
```

displays summary information from the section headers (`-h`) of `fu.o`, which is explicitly identified (`-m`) as a VAX object file in the format produced by Oasys compilers. You can list the formats available with the `-i` option. See Target Selection, for more information.

`-C`
`--demangle[=`*style*`]`

Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler. See c++filt, for more information on demangling.

`--recurse-limit`
`--no-recurse-limit`
`--recursion-limit`
`--no-recursion-limit`

Enables or disables a limit on the amount of recursion performed whilst demangling strings. Since the name mangling formats allow for an inifinite level of recursion it is possible to create strings whose decoding will exhaust the amount of stack space available on the host machine, triggering a memory fault. The limit tries to prevent this from happening by restricting recursion to 2048 levels of nesting.

The default is for this limit to be enabled, but disabling it may be necessary in order to demangle truly complicated names. Note however that if the recursion limit is disabled then stack exhaustion is possible and any bug reports about such an event will be rejected.

`-g`
`--debugging`

Display debugging information. This attempts to parse STABS debugging format information stored in the file and print it out using a C like syntax. If no STABS debuging was found this option falls back on the `-W` option to print any DWARF information in the file.

`-e`
`--debugging-tags`

Like `-g`, but the information is generated in a format compatible with ctags tool.

`-d`
`--disassemble`
`--disassemble=`*symbol*

Display the assembler mnemonics for the machine instructions from the input file. This option only disassembles those sections which are expected to contain instructions. If the optional *symbol* argument is given, then display the assembler mnemonics starting at *symbol*. If *symbol* is a function name then disassembly will stop at the end of the function, otherwise it will stop when the next symbol is encountered. If there are no matches for *symbol* then nothing will be displayed.

Note if the `--dwarf=follow-links` option has also been enabled then any symbol tables in linked debug info files will be read in and used when disassembling.

`-D`
`--disassemble-all`

Like `-d`, but disassemble the contents of all sections, not just those expected to contain instructions.

This option also has a subtle effect on the disassembly of instructions in code sections. When option `-d` is in effect objdump will assume that any symbols present in a code section occur on the boundary between instructions and it will refuse to disassemble across such a boundary. When option `-D` is in effect however this assumption is supressed. This means that it is possible for the output of `-d` and `-D` to differ if, for example, data is stored in code sections.

If the target is an ARM architecture this switch also has the effect of forcing the disassembler to decode pieces of data found in code sections as if they were instructions.

Note if the `--dwarf=follow-links` option has also been enabled then any symbol tables in linked debug info files will be read in and used when disassembling.

`--prefix-addresses`

When disassembling, print the complete address on each line. This is the older disassembly format.

`-EB`
`-EL`
`--endian={big|little}`

Specify the endianness of the object files. This only affects disassembly. This can be useful when disassembling a file format which does not describe endianness information, such as S-records.

`-f`

`--file-headers`

Display summary information from the overall header of each of the *objfile* files.

`-F`
`--file-offsets`

When disassembling sections, whenever a symbol is displayed, also display the file offset of the region of data that is about to be dumped. If zeroes are being skipped, then when disassembly resumes, tell the user how many zeroes were skipped and the file offset of the location from where the disassembly resumes. When dumping sections, display the file offset of the location from where the dump starts.

`--file-start-context`

Specify that when displaying interlisted source code/disassembly (assumes `-S`) from a file that has not yet been displayed, extend the context to the start of the file.

`-h`
`--section-headers`
`--headers`

Display summary information from the section headers of the object file.

File segments may be relocated to nonstandard addresses, for example by using the `-Ttext`, `-Tdata`, or `-Tbss` options to `ld`. However, some object file formats, such as a.out, do not store the starting address of the file segments. In those situations, although `ld` relocates the sections correctly, using '`objdump -h`' to list the file section headers cannot show the correct addresses. Instead, it shows the usual addresses, which are implicit for the target.

Note, in some cases it is possible for a section to have both the READONLY and the NOREAD attributes set. In such cases the NOREAD attribute takes precedence, but `objdump` will report both since the exact setting of the flag bits might be important.

`-H`
`--help`

Print a summary of the options to `objdump` and exit.

`-i`
`--info`

Display a list showing all architectures and object formats available for specification with `-b` or `-m`.

`-j` *`name`*
`--section=`*`name`*

Display information only for section *name*.

`-l`
`--line-numbers`

Label the display (using debugging information) with the filename and source line numbers corresponding to the object code or relocs shown. Only useful with `-d`, `-D`, or `-r`.

`-m` *`machine`*
`--architecture=`*`machine`*

Specify the architecture to use when disassembling object files. This can be useful when disassembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the `-i` option.

If the target is an ARM architecture then this switch has an additional effect. It restricts the disassembly to only those instructions supported by the architecture specified by *machine*. If it is necessary to use this switch because the input file does not contain any architecture information, but it is also desired to disassemble all the instructions use `-marm`.

`-M` *`options`*
`--disassembler-options=`*`options`*

Pass target specific information to the disassembler. Only supported on some targets. If it is necessary to specify more than one disassembler option then multiple `-M` options can be used or can be placed together into a comma separated list.

For ARC, `dsp` controls the printing of DSP instructions, `spfp` selects the printing of FPX single precision FP instructions, `dpfp` selects the printing of FPX double precision FP instructions, `quarkse_em` selects the printing of special QuarkSE-EM instructions, `fpuda` selects the printing of double precision assist instructions, `fpus` selects the printing of FPU single precision FP instructions, while `fpud` selects the printing of FPU double precision FP instructions. Additionally, one can choose to have all the immediates printed in hexadecimal using `hex`. By default, the short immediates are printed using the decimal representation, while the long immediate values are printed as hexadecimal.

`cpu=...` allows to enforce a particular ISA when disassembling instructions, overriding the `-m` value or whatever is in the ELF file. This might be useful to select ARC EM or HS ISA, because architecture is same for those and disassembler relies on private ELF header data to decide if code is for EM or HS. This option might be specified multiple times - only the latest value will be used. Valid values are same as for the assembler `-mcpu=...` option.

If the target is an ARM architecture then this switch can be used to select which register name set is used during disassembler. Specifying `-M reg-names-std` (the default) will select the register names as used in ARM's instruction set documentation, but with register 13 called 'sp', register 14 called 'lr' and register 15 called 'pc'. Specifying `-M reg-names-apcs` will select the name set used by the ARM Procedure Call Standard, whilst specifying `-M reg-names-raw` will just use '`r`' followed by the register number.

There are also two variants on the APCS register naming scheme enabled by `-M reg-names-atpcs` and `-M reg-names-special-atpcs` which use the ARM/Thumb Procedure Call Standard naming conventions. (Either with the normal register names or the special register names).

This option can also be used for ARM architectures to force the disassembler to interpret all instructions as Thumb instructions by using the switch `--disassembler-options=force-thumb`. This can be useful when attempting to disassemble thumb code produced by other compilers.

For AArch64 targets this switch can be used to set whether instructions are disassembled as the most general instruction using the `-M no-aliases` option or whether instruction notes should be generated as comments in the disasssembly using `-M notes`.

For the x86, some of the options duplicate functions of the `-m` switch, but allow finer grained control. Multiple selections from the following may be specified as a comma separated string.

`x86-64`
`i386`
`i8086`

Select disassembly for the given architecture.

`intel`
`att`

Select between intel syntax mode and AT&T syntax mode.

`amd64`
`intel64`

Select between AMD64 ISA and Intel64 ISA.

`intel-mnemonic`
`att-mnemonic`

Select between intel mnemonic mode and AT&T mnemonic mode. Note: `intel-mnemonic` implies `intel` and `att-mnemonic` implies `att`.

`addr64`
`addr32`
`addr16`
`data32`
`data16`

Specify the default address size and operand size. These five options will be overridden if `x86-64`, `i386` or `i8086` appear later in the option string.

`suffix`

When in AT&T mode, instructs the disassembler to print a mnemonic suffix even when the suffix could be inferred by the operands.

For PowerPC, the `-M` argument `raw` selects disasssembly of hardware insns rather than aliases. For example, you will see `rlwinm` rather than `clrlwi`, and `addi` rather than `li`. All of the `-m` arguments for `gas` that select a CPU are supported. These are: `403`, `405`, `440`, `464`, `476`, `601`, `603`, `604`, `620`, `7400`, `7410`, `7450`, `7455`, `750cl`, `821`, `850`, `860`, `a2`, `booke`, `booke32`, `cell`, `com`, `e200z4`, `e300`, `e500`, `e500mc`, `e500mc64`, `e500x2`, `e5500`, `e6500`, `efs`, `power4`, `power5`, `power6`, `power7`, `power8`, `power9`, `ppc`, `ppc32`, `ppc64`, `ppc64bridge`, `ppcps`, `pwr`, `pwr2`, `pwr4`, `pwr5`, `pwr5x`, `pwr6`, `pwr7`, `pwr8`, `pwr9`, `pwrx`, `titan`, and `vle`. `32` and `64` modify the default or a prior CPU selection, disabling and enabling 64-bit insns respectively. In addition, `altivec`, `any`, `htm`, `vsx`, and `spe` add capabilities to a previous *or later* CPU selection. `any` will disassemble any opcode known to binutils, but in cases where an opcode has two different meanings or different arguments, you may not see the disassembly you expect. If you disassemble without giving a CPU selection, a default will be chosen from information gleaned by BFD from the object files headers, but the result again may not be as you expect.

For MIPS, this option controls the printing of instruction mnemonic names and register names in disassembled instructions. Multiple selections from the following may be specified as a comma separated string, and invalid options are ignored:

`no-aliases`

Print the 'raw' instruction mnemonic instead of some pseudo instruction mnemonic. I.e., print 'daddu' or 'or' instead of 'move', 'sll' instead of 'nop', etc.

`msa`

Disassemble MSA instructions.

`virt`

Disassemble the virtualization ASE instructions.

`xpa`

Disassemble the eXtended Physical Address (XPA) ASE instructions.

`gpr-names=`*`ABI`*

Print GPR (general-purpose register) names as appropriate for the specified ABI. By default, GPR names are selected according to the ABI of the binary being disassembled.

`fpr-names=`*`ABI`*

Print FPR (floating-point register) names as appropriate for the specified ABI. By default, FPR numbers are printed rather than names.

`cp0-names=`*`ARCH`*

Print CP0 (system control coprocessor; coprocessor 0) register names as appropriate for the CPU or architecture specified by *ARCH*. By default, CP0 register names are selected according to the architecture and CPU of the binary being disassembled.

`hwr-names=`*`ARCH`*

Print HWR (hardware register, used by the `rdhwr` instruction) names as appropriate for the CPU or architecture specified by *ARCH*. By default, HWR names are selected according to the architecture and CPU of the binary being disassembled.

`reg-names=`*`ABI`*

Print GPR and FPR names as appropriate for the selected ABI.

`reg-names=`*`ARCH`*

Print CPU-specific register names (CP0 register and HWR names) as appropriate for the selected CPU or architecture.

For any of the options listed above, *ABI* or *ARCH* may be specified as '`numeric`' to have numbers printed rather than names, for the selected types of registers. You can list the available values of *ABI* and *ARCH* using the `--help` option.

For VAX, you can specify function entry addresses with `-M entry:0xf00ba`. You can use this multiple times to properly disassemble VAX binary files that don't contain symbol tables (like ROM dumps). In these cases, the function entry mask would otherwise be decoded as VAX instructions, which would probably lead the rest of the function being wrongly disassembled.

`-p`
`--private-headers`

Print information that is specific to the object file format. The exact information printed depends upon the object file format. For some object file formats, no additional information is printed.

`-P `*`options`*
`--private=`*`options`*

Print information that is specific to the object file format. The argument *options* is a comma separated list that depends on the format (the lists of options is displayed with the help).

For XCOFF, the available options are:

```
header
aout
sections
syms
relocs
lineno,
loader
except
typchk
traceback
toc
ldinfo
```

Not all object formats support this option. In particular the ELF format does not use it.

`-r`
`--reloc`

Print the relocation entries of the file. If used with `-d` or `-D`, the relocations are printed interspersed with the disassembly.

`-R`
`--dynamic-reloc`

Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. As for `-r`, if used with `-d` or `-D`, the relocations are printed interspersed with the disassembly.

`-s`
`--full-contents`

Display the full contents of any sections requested. By default all non-empty sections are displayed.

`-S`
`--source`

Display source code intermixed with disassembly, if possible. Implies `-d`.

`--source-comment[=`*`txt`*`]`

Like the `-S` option, but all source code lines are displayed with a prefix of *txt*. Typically *txt* will be a comment string which can be used to distinguish the assembler code from the source code. If *txt* is not provided then a default string of *"# "* (hash followed by a space), will be used.

`--prefix=`*`prefix`*

Specify *prefix* to add to the absolute paths when used with `-S`.

`--prefix-strip=`*`level`*

Indicate how many initial directory names to strip off the hardwired absolute paths. It has no effect without `--prefix=`*prefix*.

`--show-raw-insn`

When disassembling instructions, print the instruction in hex as well as in symbolic form. This is the default except when `--prefix-addresses` is used.

`--no-show-raw-insn`

When disassembling instructions, do not print the instruction bytes. This is the default when `--prefix-addresses` is used.

`--insn-width=`*`width`*

Display *width* bytes on a single line when disassembling instructions.

`-W[lLiaprmfFsoRtUuTgAckK]`
`--dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges,=pubtypes,=trace_info,=trace_abbrev,=trace_aranges,=gdb_index,=addr,=cu_index,=links,=follow-links]`

Displays the contents of the DWARF debug sections in the file, if any are present. Compressed debug sections are automatically decompressed (temporarily) before they are displayed. If one or more of the optional letters or words follows the switch then only those type(s) of data will be dumped. The letters and words refer to the following information:

`a`
`=abbrev`

Displays the contents of the '`.debug_abbrev`' section.

`A`
`=addr`

Displays the contents of the '`.debug_addr`' section.

`c`
`=cu_index`

Displays the contents of the '`.debug_cu_index`' and/or '`.debug_tu_index`' sections.

`f`
`=frames`

Display the raw contents of a '`.debug_frame`' section.

`F`
`=frame-interp`

Display the interpreted contents of a '`.debug_frame`' section.

`g`
`=gdb_index`

Displays the contents of the '`.gdb_index`' and/or '`.debug_names`' sections.

`i`

`=info`

Displays the contents of the '`.debug_info`' section. Note: the output from this option can also be restricted by the use of the `--dwarf-depth` and `--dwarf-start` options.

`k`
`=links`

Displays the contents of the '`.gnu_debuglink`' and/or '`.gnu_debugaltlink`' sections. Also displays any links to separate dwarf object files (dwo), if they are specified by the DW_AT_GNU_dwo_name or DW_AT_dwo_name attributes in the '`.debug_info`' section.

`K`
`=follow-links`

Display the contents of any selected debug sections that are found in linked, separate debug info file(s). This can result in multiple versions of the same debug section being displayed if it exists in more than one file.

In addition, when displaying DWARF attributes, if a form is found that references the separate debug info file, then the referenced contents will also be displayed.

`l`
`=rawline`

Displays the contents of the '`.debug_line`' section in a raw format.

`L`
`=decodedline`

Displays the interpreted contents of the '`.debug_line`' section.

`m`
`=macro`

Displays the contents of the '`.debug_macro`' and/or '`.debug_macinfo`' sections.

`o`
`=loc`

Displays the contents of the '`.debug_loc`' and/or '`.debug_loclists`' sections.

`p`
`=pubnames`

Displays the contents of the '`.debug_pubnames`' and/or '`.debug_gnu_pubnames`' sections.

`r`
`=aranges`

Displays the contents of the '`.debug_aranges`' section.

`R`
`=Ranges`

Displays the contents of the '`.debug_ranges`' and/or '`.debug_rnglists`' sections.

`s`
`=str`

Displays the contents of the '`.debug_str`', '`.debug_line_str`' and/or '`.debug_str_offsets`' sections.

`t`
`=pubtype`

Displays the contents of the '`.debug_pubtypes`' and/or '`.debug_gnu_pubtypes`' sections.

`T`
`=trace_aranges`

Displays the contents of the '`.trace_aranges`' section.

`u`
`=trace_abbrev`

Displays the contents of the '`.trace_abbrev`' section.

`U`
`=trace_info`

Displays the contents of the '`.trace_info`' section.

Note: displaying the contents of '`.debug_static_funcs`', '`.debug_static_vars`' and '`debug_weaknames`' sections is not currently supported.

`--dwarf-depth=`*`n`*

Limit the dump of the `.debug_info` section to *n* children. This is only useful with `--debug-dump=info`. The default is to print all DIEs; the special value 0 for *n* will also have this effect.

With a non-zero value for *n*, DIEs at or deeper than *n* levels will not be printed. The range for *n* is zero-based.

`--dwarf-start=`*`n`*

Print only DIEs beginning with the DIE numbered *n*. This is only useful with `--debug-dump=info`.

If specified, this option will suppress printing of any header information and all DIEs before the DIE numbered *n*. Only siblings and children of the specified DIE will be printed.

This can be used in conjunction with `--dwarf-depth`.

`--dwarf-check`

Enable additional checks for consistency of Dwarf information.

`--ctf=`*`section`*

Display the contents of the specified CTF section. CTF sections themselves contain many subsections, all of which are displayed in order.

`--ctf-parent=`*`section`*

Specify the name of another section from which the CTF file can inherit types.

`-G`
`--stabs`

Display the full contents of any sections requested. Display the contents of the .stab and .stab.index and .stab.excl sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which `.stab` debugging symbol-table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the `--syms` output.

`--start-address=`*`address`*

Start displaying data at the specified address. This affects the output of the `-d`, `-r` and `-s` options.

`--stop-address=`*`address`*

Stop displaying data at the specified address. This affects the output of the `-d`, `-r` and `-s` options.

`-t`
`--syms`

Print the symbol table entries of the file. This is similar to the information provided by the '`nm`' program, although the display format is different. The format of the output depends upon the format of the file being dumped, but there are two main types. One looks like this:

```
[  4](sec  3)(fl 0x00)(ty   0)(scl   3) (nx 1) 0x00000000 .bss
[  6](sec  1)(fl 0x00)(ty   0)(scl   2) (nx 0) 0x00000000 fred
```

where the number inside the square brackets is the number of the entry in the symbol table, the *sec* number is the section number, the *fl* value are the symbol's flag bits, the *ty* number is the symbol's type, the *scl* number is the symbol's storage class and the *nx* value is the number of auxilary entries associated with the symbol. The last two fields are the symbol's value and its name.

The other common output format, usually seen with ELF based files, looks like this:

```
00000000 l    d  .bss   00000000 .bss
00000000 g       .text  00000000 fred
```

Here the first number is the symbol's value (sometimes refered to as its address). The next field is actually a set of characters and spaces indicating the flag bits that are set on the symbol. These characters are described below. Next is the section with which the symbol is associated or **ABS** if the section is absolute (ie not connected with any section), or **UND** if the section is referenced in the file being dumped, but not defined there.

After the section name comes another field, a number, which for common symbols is the alignment and for other symbol is the size. Finally the symbol's name is displayed.

The flag characters are divided into 7 groups as follows:

`l`
`g`
`u`
`!`

The symbol is a local (l), global (g), unique global (u), neither global nor local (a space) or both global and local (!). A symbol can be neither local or global for a variety of reasons, e.g., because it is used for debugging, but it is probably an indication of a bug if it is ever both local and global. Unique global symbols are a GNU extension to the standard set of ELF symbol bindings. For such a symbol the dynamic linker will make sure that in the entire process there is just one symbol with this name and type in use.

`w`

The symbol is weak (w) or strong (a space).

`C`

The symbol denotes a constructor (C) or an ordinary symbol (a space).

`W`

The symbol is a warning (W) or a normal symbol (a space). A warning symbol's name is a message to be displayed if the symbol following the warning symbol is ever referenced.

`I`
`i`

The symbol is an indirect reference to another symbol (I), a function to be evaluated during reloc processing (i) or a normal symbol (a space).

`d`
`D`

The symbol is a debugging symbol (d) or a dynamic symbol (D) or a normal symbol (a space).

`F`
`f`
`O`

The symbol is the name of a function (F) or a file (f) or an object (O) or just a normal symbol (a space).

`-T`
`--dynamic-syms`

Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the '`nm`' program when given the `-D` (`--dynamic`) option.

The output format is similar to that produced by the `--syms` option, except that an extra field is inserted before the symbol's name, giving the version information associated with the symbol. If the version is the default version to be used when resolving unversioned references to the symbol then it's displayed as is, otherwise it's put into parentheses.

`--special-syms`

When displaying symbols include those which the target considers to be special in some way and which would not normally be of interest to the user.

`-V`
`--version`

Print the version number of `objdump` and exit.

`-x`
`--all-headers`

Display all available header information, including the symbol table and relocation entries. Using `-x` is equivalent to specifying all of `-a -f -h -p -r -t`.

`-w`
`--wide`

Format some lines for output devices that have more than 80 columns. Also do not truncate symbol names when they are displayed.

`-z`
`--disassemble-zeroes`

Normally the disassembly output will skip blocks of zeroes. This option directs the disassembler to disassemble those blocks, just like any other data.

Next: ranlib, Previous: objcopy, Up: Top   [Contents][Index]