

libbfd

The Binary File Descriptor Library

First Edition—BFD version < 3.0 % Since no product is stable before version 3.0 :-)

Original Document Created: April 1991

Steve Chamberlain
Cygnus Support

Free Software Foundation
sac@www.gnu.org
BFD, 1.5
T_EXinfo 2022-11-07.17

Copyright © 1991-2023 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Table of Contents

1	Introduction	1
1.1	History	1
1.2	How To Use BFD	1
1.3	What BFD Version 2 Can Do	2
1.3.1	Information Loss	2
1.3.2	The BFD canonical object-file format	3
2	BFD Front End	5
2.1	typedef bfd	5
2.2	Error reporting	12
2.2.1	Type bfd_error_type	12
2.2.1.1	bfd_get_error	12
2.2.1.2	bfd_set_error	12
2.2.1.3	bfd_set_input_error	13
2.2.1.4	bfd_errmsg	13
2.2.1.5	bfd_perror	13
2.2.1.6	bfd_asprintf	13
2.2.2	BFD error handler	13
2.2.2.1	_bfd_error_handler	13
2.2.2.2	bfd_set_error_handler	14
2.2.2.3	_bfd_set_error_handler_caching	14
2.2.2.4	bfd_set_error_program_name	14
2.2.2.5	_bfd_get_error_program_name	14
2.2.3	BFD assert handler	14
2.2.3.1	bfd_set_assert_handler	14
2.2.3.2	bfd_init	14
2.3	Miscellaneous	15
2.3.1	Miscellaneous functions	15
2.3.1.1	bfd_get_reloc_upper_bound	15
2.3.1.2	bfd_canonicalize_reloc	15
2.3.1.3	bfd_set_reloc	15
2.3.1.4	bfd_set_file_flags	15
2.3.1.5	bfd_get_arch_size	15
2.3.1.6	bfd_get_sign_extend_vma	16
2.3.1.7	bfd_set_start_address	16
2.3.1.8	bfd_get_gp_size	16
2.3.1.9	bfd_set_gp_size	16
2.3.1.10	bfd_set_gp_value	16
2.3.1.11	bfd_scan_vma	16
2.3.1.12	bfd_copy_private_header_data	17
2.3.1.13	bfd_copy_private_bfd_data	17
2.3.1.14	bfd_set_private_flags	17
2.3.1.15	Other functions	17

2.3.1.16	bfd_get_relocated_section_contents	20
2.3.1.17	bfd_record_phdr	20
2.3.1.18	bfd_sprintf_vma	20
2.3.1.19	bfd_alt_mach_code	20
2.3.1.20	bfd_emul_get_maxpagesize	20
2.3.1.21	bfd_emul_get_commonpagesize	20
2.3.1.22	bfd_demangle	20
2.3.1.23	struct bfd_iovec	21
2.3.1.24	bfd_bread	21
2.3.1.25	bfd_bwrite	22
2.3.1.26	bfd_tell	22
2.3.1.27	bfd_flush	22
2.3.1.28	bfd_stat	22
2.3.1.29	bfd_seek	22
2.3.1.30	bfd_get_mtime	22
2.3.1.31	bfd_get_size	22
2.3.1.32	bfd_get_file_size	23
2.3.1.33	bfd_mmap	23
2.4	Memory Usage	23
2.5	Sections	23
2.5.1	Section input	23
2.5.2	Section output	24
2.5.3	Link orders	24
2.5.4	typedef asection	25
2.5.5	Section prototypes	33
2.5.5.1	bfd_section_list_clear	33
2.5.5.2	bfd_get_section_by_name	34
2.5.5.3	bfd_get_next_section_by_name	34
2.5.5.4	bfd_get_linker_section	34
2.5.5.5	bfd_get_section_by_name_if	34
2.5.5.6	bfd_get_unique_section_name	34
2.5.5.7	bfd_make_section_old_way	34
2.5.5.8	bfd_make_section_anyway_with_flags	35
2.5.5.9	bfd_make_section_anyway	35
2.5.5.10	bfd_make_section_with_flags	35
2.5.5.11	bfd_make_section	35
2.5.5.12	bfd_set_section_flags	35
2.5.5.13	bfd_rename_section	36
2.5.5.14	bfd_map_over_sections	36
2.5.5.15	bfd_sections_find_if	36
2.5.5.16	bfd_set_section_size	36
2.5.5.17	bfd_set_section_contents	37
2.5.5.18	bfd_get_section_contents	37
2.5.5.19	bfd_malloc_and_get_section	37
2.5.5.20	bfd_copy_private_section_data	37
2.5.5.21	bfd_generic_is_group_section	38
2.5.5.22	bfd_generic_group_name	38

2.5.5.23	<code>bfd_generic_discard_group</code>	38
2.5.5.24	<code>_bfd_section_size_insane</code>	38
2.6	Symbols.....	38
2.6.1	Reading symbols.....	38
2.6.2	Writing symbols.....	39
2.6.3	Mini Symbols.....	40
2.6.4	<code>typedef asymbol</code>	40
2.6.5	Symbol handling functions.....	43
2.6.5.1	<code>bfd_get_syntab_upper_bound</code>	43
2.6.5.2	<code>bfd_is_local_label</code>	44
2.6.5.3	<code>bfd_is_local_label_name</code>	44
2.6.5.4	<code>bfd_is_target_special_symbol</code>	44
2.6.5.5	<code>bfd_canonicalize_syntab</code>	44
2.6.5.6	<code>bfd_set_syntab</code>	44
2.6.5.7	<code>bfd_print_symbol_vandf</code>	45
2.6.5.8	<code>bfd_make_empty_symbol</code>	45
2.6.5.9	<code>_bfd_generic_make_empty_symbol</code>	45
2.6.5.10	<code>bfd_make_debug_symbol</code>	45
2.6.5.11	<code>bfd_decode_symclass</code>	45
2.6.5.12	<code>bfd_is_undefined_symclass</code>	45
2.6.5.13	<code>bfd_symbol_info</code>	45
2.6.5.14	<code>bfd_copy_private_symbol_data</code>	46
2.7	Archives.....	46
2.7.1	Archive functions.....	47
2.7.1.1	<code>bfd_get_next_mapent</code>	47
2.7.1.2	<code>bfd_set_archive_head</code>	47
2.7.1.3	<code>bfd_openr_next_archived_file</code>	47
2.8	File formats.....	47
2.8.1	File format functions.....	47
2.8.1.1	<code>bfd_check_format</code>	48
2.8.1.2	<code>bfd_check_format_matches</code>	48
2.8.1.3	<code>bfd_set_format</code>	48
2.8.1.4	<code>bfd_format_string</code>	48
2.9	Relocations.....	48
2.9.1	<code>typedef arelent</code>	49
2.9.1.1	<code>enum complain_overflow</code>	51
2.9.1.2	<code>reloc_howto_type</code>	51
2.9.1.3	The HOWTO Macro.....	53
2.9.1.4	<code>arelent_chain</code>	54
2.9.1.5	<code>bfd_check_overflow</code>	54
2.9.1.6	<code>bfd_reloc_offset_in_range</code>	54
2.9.1.7	<code>bfd_perform_relocation</code>	55
2.9.1.8	<code>bfd_install_relocation</code>	55
2.9.2	The howto manager.....	55
2.9.2.1	<code>bfd_reloc_code_real_type</code>	55
2.9.2.2	<code>bfd_reloc_type_lookup</code>	136
2.9.2.3	<code>bfd_default_reloc_type_lookup</code>	136

2.9.2.4	bfd_get_reloc_code_name	136
2.9.2.5	bfd_generic_relax_section	136
2.9.2.6	bfd_generic_gc_sections	136
2.9.2.7	bfd_generic_lookup_section_flags	136
2.9.2.8	bfd_generic_merge_sections	136
2.9.2.9	bfd_generic_get_relocated_section_contents	137
2.9.2.10	_bfd_generic_set_reloc	137
2.9.2.11	_bfd_unrecognized_reloc	137
2.10	Core files	137
2.10.1	Core file functions	137
2.10.1.1	bfd_core_file_failing_command	137
2.10.1.2	bfd_core_file_failing_signal	137
2.10.1.3	bfd_core_file_pid	137
2.10.1.4	core_file_matches_executable_p	137
2.10.1.5	generic_core_file_matches_executable_p	138
2.11	Targets	138
2.11.1	bfd_target	138
2.11.1.1	_bfd_per_xvec_warn	150
2.11.1.2	bfd_set_default_target	150
2.11.1.3	bfd_find_target	150
2.11.1.4	bfd_get_target_info	150
2.11.1.5	bfd_target_list	151
2.11.1.6	bfd_iterate_over_targets	151
2.11.1.7	bfd_flavour_name	151
2.12	Architectures	151
2.12.1	bfd_architecture	151
2.12.2	bfd_arch_info	162
2.12.2.1	bfd_printable_name	163
2.12.2.2	bfd_scan_arch	163
2.12.2.3	bfd_arch_list	163
2.12.2.4	bfd_arch_get_compatible	164
2.12.2.5	bfd_default_arch_struct	164
2.12.2.6	bfd_set_arch_info	164
2.12.2.7	bfd_default_set_arch_mach	164
2.12.2.8	bfd_get_arch	164
2.12.2.9	bfd_get_mach	164
2.12.2.10	bfd_arch_bits_per_byte	164
2.12.2.11	bfd_arch_bits_per_address	164
2.12.2.12	bfd_default_compatible	165
2.12.2.13	bfd_default_scan	165
2.12.2.14	bfd_get_arch_info	165
2.12.2.15	bfd_lookup_arch	165
2.12.2.16	bfd_printable_arch_mach	165
2.12.2.17	bfd_octets_per_byte	165
2.12.2.18	bfd_arch_mach_octets_per_byte	165
2.12.2.19	bfd_arch_default_fill	166
2.13	Opening and closing BFDs	166

2.13.1	Functions for opening and closing.....	166
2.13.1.1	_bfd_new_bfd.....	166
2.13.1.2	_bfd_new_bfd_contained_in.....	166
2.13.1.3	_bfd_free_cached_info	166
2.13.1.4	bfd_fopen	166
2.13.1.5	bfd_openr	166
2.13.1.6	bfd_fdopenr	167
2.13.1.7	bfd_fdopenw.....	167
2.13.1.8	bfd_openstreamr	167
2.13.1.9	bfd_openr_iovec	168
2.13.1.10	bfd_openw	168
2.13.1.11	bfd_elf_bfd_from_remote_memory	168
2.13.1.12	bfd_close	169
2.13.1.13	bfd_close_all_done.....	169
2.13.1.14	bfd_create.....	169
2.13.1.15	bfd_make_writable.....	169
2.13.1.16	bfd_make_readable.....	170
2.13.1.17	bfd_calc_gnu_debuglink_crc32.....	170
2.13.1.18	bfd_get_debug_link_info.....	170
2.13.1.19	bfd_get_alt_debug_link_info.....	170
2.13.1.20	bfd_follow_gnu_debuglink.....	170
2.13.1.21	bfd_follow_gnu_debugaltlink.....	171
2.13.1.22	bfd_create_gnu_debuglink_section.....	171
2.13.1.23	bfd_fill_in_gnu_debuglink_section.....	171
2.13.1.24	bfd_follow_build_id_debuglink.....	171
2.13.1.25	bfd_set_filename	172
2.14	File caching.....	172
2.14.1	Caching functions	172
2.14.1.1	bfd_cache_init.....	172
2.14.1.2	bfd_cache_close	172
2.14.1.3	bfd_cache_close_all.....	172
2.14.1.4	bfd_open_file.....	172
2.15	Linker Functions	173
2.15.1	Creating a linker hash table	173
2.15.2	Adding symbols to the hash table.....	173
2.15.2.1	Differing file formats.....	174
2.15.2.2	Adding symbols from an object file	174
2.15.2.3	Adding symbols from an archive	175
2.15.3	Performing the final link.....	175
2.15.3.1	Information provided by the linker.....	176
2.15.3.2	Relocating the section contents	176
2.15.3.3	Writing the symbol table	176
2.15.3.4	bfd_link_split_section.....	177
2.15.3.5	bfd_section_already_linked	177
2.15.3.6	bfd_generic_define_common_symbol.....	177
2.15.3.7	_bfd_generic_link_hide_symbol.....	178
2.15.3.8	bfd_generic_define_start_stop.....	178

2.15.3.9	bfd_find_version_for_sym.....	178
2.15.3.10	bfd_hide_sym_by_version.....	178
2.15.3.11	bfd_link_check_relocs.....	178
2.15.3.12	_bfd_generic_link_check_relocs.....	178
2.15.3.13	bfd_merge_private_bfd_data.....	179
2.15.3.14	_bfd_generic_verify_endian_match.....	179
2.16	Hash Tables.....	179
2.16.1	Creating and freeing a hash table.....	179
2.16.2	Looking up or entering a string.....	180
2.16.3	Traversing a hash table.....	180
2.16.4	Deriving a new hash table type.....	180
2.16.4.1	Define the derived structures.....	180
2.16.4.2	Write the derived creation routine.....	181
2.16.4.3	Write other derived routines.....	182
2.16.4.4	bfd_hash_table_init_n.....	182
2.16.4.5	bfd_hash_table_init.....	182
2.16.4.6	bfd_hash_table_free.....	182
2.16.4.7	bfd_hash_lookup.....	183
2.16.4.8	bfd_hash_insert.....	183
2.16.4.9	bfd_hash_rename.....	183
2.16.4.10	bfd_hash_replace.....	183
2.16.4.11	bfd_hash_allocate.....	183
2.16.4.12	bfd_hash_newfunc.....	183
2.16.4.13	bfd_hash_traverse.....	183
2.16.4.14	bfd_hash_set_default_size.....	183
2.16.4.15	_bfd_stringtab_init.....	183
2.16.4.16	_bfd_xcoff_stringtab_init.....	184
2.16.4.17	_bfd_stringtab_free.....	184
2.16.4.18	_bfd_stringtab_add.....	184
2.16.4.19	_bfd_stringtab_size.....	184
2.16.4.20	_bfd_stringtab_emit.....	184
3	BFD back ends.....	185
3.1	What to Put Where.....	185
3.2	a.out backends.....	185
3.2.1	Relocations.....	186
3.2.2	Internal entry points.....	186
3.2.2.1	aout_size_swap_exec_header_in.....	186
3.2.2.2	aout_size_swap_exec_header_out.....	186
3.2.2.3	aout_size_some_aout_object_p.....	186
3.2.2.4	aout_size_mkobject.....	187
3.2.2.5	aout_size_machine_type.....	187
3.2.2.6	aout_size_set_arch_mach.....	187
3.2.2.7	aout_size_new_section_hook.....	187
3.3	coff backends.....	187
3.3.1	Porting to a new version of coff.....	187

3.3.2	How the coff backend works	188
3.3.2.1	File layout.....	188
3.3.2.2	Coff long section names.....	188
3.3.2.3	Bit twiddling.....	189
3.3.2.4	Symbol reading.....	189
3.3.2.5	Symbol writing.....	190
3.3.2.6	coff_symbol_type.....	191
3.3.2.7	bfd_coff_backend_data.....	192
3.3.2.8	Writing relocations.....	195
3.3.2.9	Reading linenumbers.....	195
3.3.2.10	Reading relocations.....	195
3.4	ELF backends.....	196
3.5	mmo backend.....	196
3.5.1	File layout.....	196
3.5.2	Symbol table format.....	198
3.5.3	mmo section mapping.....	200
	BFD Index	210

1 Introduction

BFD is a package which allows applications to use the same routines to operate on object files whatever the object file format. A new object file format can be supported simply by creating a new BFD back end and adding it to the library.

BFD is split into two parts: the front end, and the back ends (one for each object file format).

- The front end of BFD provides the interface to the user. It manages memory and various canonical data structures. The front end also decides which back end to use and when to call back end routines.
- The back ends provide BFD its view of the real world. Each back end provides a set of calls which the BFD front end can use to maintain its canonical form. The back ends also may keep around information for their own use, for greater efficiency.

1.1 History

One spur behind BFD was the desire, on the part of the GNU 960 team at Intel Oregon, for interoperability of applications on their COFF and b.out file formats. Cygnus was providing GNU support for the team, and was contracted to provide the required functionality.

The name came from a conversation David Wallace was having with Richard Stallman about the library: RMS said that it would be quite hard—David said “BFD”. Stallman was right, but the name stuck.

At the same time, Ready Systems wanted much the same thing, but for different object file formats: IEEE-695, Oasys, Srecords, a.out and 68k coff.

BFD was first implemented by members of Cygnus Support; Steve Chamberlain (sac@cygnus.com), John Gilmore (gnu@cygnus.com), K. Richard Pixley (rich@cygnus.com) and David Henkel-Wallace (gumby@cygnus.com).

1.2 How To Use BFD

To use the library, include `bfd.h` and link with `libbfd.a`.

BFD provides a common interface to the parts of an object file for a calling application.

When an application successfully opens a target file (object, archive, or whatever), a pointer to an internal structure is returned. This pointer points to a structure called `bfd`, described in `bfd.h`. Our convention is to call this pointer a BFD, and instances of it within code `abfd`. All operations on the target object file are applied as methods to the BFD. The mapping is defined within `bfd.h` in a set of macros, all beginning with `'bfd_'` to reduce namespace pollution.

For example, this sequence does what you would probably expect: return the number of sections in an object file attached to a BFD `abfd`.

```
#include "bfd.h"

unsigned int number_of_sections (abfd)
bfd *abfd;
{
```

```
    return bfd_count_sections (abfd);  
}
```

The abstraction used within BFD is that an object file has:

- a header,
- a number of sections containing raw data (see Section 2.5 [Sections], page 23),
- a set of relocations (see Section 2.9 [Relocations], page 48), and
- some symbol information (see Section 2.6 [Symbols], page 38).

Also, BFDs opened for archives have the additional attribute of an index and contain subordinate BFDs. This approach is fine for a.out and coff, but loses efficiency when applied to formats such as S-records and IEEE-695.

1.3 What BFD Version 2 Can Do

When an object file is opened, BFD subroutines automatically determine the format of the input object file. They then build a descriptor in memory with pointers to routines that will be used to access elements of the object file's data structures.

As different information from the object files is required, BFD reads from different sections of the file and processes them. For example, a very common operation for the linker is processing symbol tables. Each BFD back end provides a routine for converting between the object file's representation of symbols and an internal canonical format. When the linker asks for the symbol table of an object file, it calls through a memory pointer to the routine from the relevant BFD back end which reads and converts the table into a canonical form. The linker then operates upon the canonical form. When the link is finished and the linker writes the output file's symbol table, another BFD back end routine is called to take the newly created symbol table and convert it into the chosen output format.

1.3.1 Information Loss

Information can be lost during output. The output formats supported by BFD do not provide identical facilities, and information which can be described in one form has nowhere to go in another format. One example of this is alignment information in b.out. There is nowhere in an a.out format file to store alignment information on the contained data, so when a file is linked from b.out and an a.out image is produced, alignment information will not propagate to the output file. (The linker will still use the alignment information internally, so the link is performed correctly).

Another example is COFF section names. COFF files may contain an unlimited number of sections, each one with a textual section name. If the target of the link is a format which does not have many sections (e.g., a.out) or has sections without names (e.g., the Oasys format), the link cannot be done simply. You can circumvent this problem by describing the desired input-to-output section mapping with the linker command language.

Information can be lost during canonicalization. The BFD internal canonical form of the external formats is not exhaustive; there are structures in input formats for which there is no direct representation internally. This means that the BFD back ends cannot maintain all possible data richness through the transformation between external to internal and back to external formats.

This limitation is only a problem when an application reads one format and writes another. Each BFD back end is responsible for maintaining as much data as possible, and the internal BFD canonical form has structures which are opaque to the BFD core, and exported only to the back ends. When a file is read in one format, the canonical form is generated for BFD and the application. At the same time, the back end saves away any information which may otherwise be lost. If the data is then written back in the same format, the back end routine will be able to use the canonical form provided by the BFD core as well as the information it prepared earlier. Since there is a great deal of commonality between back ends, there is no information lost when linking or copying big endian COFF to little endian COFF, or `a.out` to `b.out`. When a mixture of formats is linked, the information is only lost from the files whose format differs from the destination.

1.3.2 The BFD canonical object-file format

The greatest potential for loss of information occurs when there is the least overlap between the information provided by the source format, that stored by the canonical format, and that needed by the destination format. A brief description of the canonical form may help you understand which kinds of data you can count on preserving across conversions.

files Information stored on a per-file basis includes target machine architecture, particular implementation format type, a demand pageable bit, and a write protected bit. Information like Unix magic numbers is not stored here—only the magic numbers' meaning, so a `ZMAGIC` file would have both the demand pageable bit and the write protected text bit set. The byte order of the target is stored on a per-file basis, so that big- and little-endian object files may be used with one another.

sections Each section in the input file contains the name of the section, the section's original address in the object file, size and alignment information, various flags, and pointers into other BFD data structures.

symbols Each symbol contains a pointer to the information for the object file which originally defined it, its name, its value, and various flag bits. When a BFD back end reads in a symbol table, it relocates all symbols to make them relative to the base of the section where they were defined. Doing this ensures that each symbol points to its containing section. Each symbol also has a varying amount of hidden private data for the BFD back end. Since the symbol points to the original file, the private data format for that symbol is accessible. `ld` can operate on a collection of symbols of wildly different formats without problems. Normal global and simple local symbols are maintained on output, so an output file (no matter its format) will retain symbols pointing to functions and to global, static, and common variables. Some symbol information is not worth retaining; in `a.out`, type information is stored in the symbol table as long symbol names. This information would be useless to most COFF debuggers; the linker has command-line switches to allow users to throw it away.

There is one word of type information within the symbol, so if the format supports symbol type information within symbols (for example, COFF, Oasys) and the type is simple enough to fit within one word (nearly everything but aggregates), the information will be preserved.

relocation level

Each canonical BFD relocation record contains a pointer to the symbol to relocate to, the offset of the data to relocate, the section the data is in, and a pointer to a relocation type descriptor. Relocation is performed by passing messages through the relocation type descriptor and the symbol pointer. Therefore, relocations can be performed on output data using a relocation method that is only available in one of the input formats. For instance, Oasys provides a byte relocation format. A relocation record requesting this relocation type would point indirectly to a routine to perform this, so the relocation may be performed on a byte being written to a 68k COFF file, even though 68k COFF has no such relocation type.

line numbers

Object formats can contain, for debugging purposes, some form of mapping between symbols, source line numbers, and addresses in the output file. These addresses have to be relocated along with the symbol information. Each symbol with an associated list of line number records points to the first record of the list. The head of a line number list consists of a pointer to the symbol, which allows finding out the address of the function whose line number is being described. The rest of the list is made up of pairs: offsets into the section and line numbers. Any format which can simply derive this information can pass it successfully between formats.

2 BFD Front End

2.1 typedef bfd

A BFD has type `bfd`; objects of this type are the cornerstone of any application using BFD. Using BFD consists of making references though the BFD and to data in the BFD.

Here is the structure that defines the type `bfd`. It contains the major data about the file and pointers to the rest of the data.

```

struct bfd
{
    /* The filename the application opened the BFD with.  */
    const char *filename;

    /* A pointer to the target jump table.  */
    const struct bfd_target *xvec;

    /* The IOSTREAM, and corresponding IO vector that provide access
       to the file backing the BFD.  */
    void *iostream;
    const struct bfd_iovec *iovec;

    /* The caching routines use these to maintain a
       least-recently-used list of BFDs.  */
    struct bfd *lru_prev, *lru_next;

    /* Track current file position (or current buffer offset for
       in-memory BFDs).  When a file is closed by the caching routines,
       BFD retains state information on the file here.  */
    ufile_ptr where;

    /* File modified time, if mtime_set is TRUE.  */
    long mtime;

    /* A unique identifier of the BFD  */
    unsigned int id;

    /* Format_specific flags.  */
    flagword flags;

    /* Values that may appear in the flags field of a BFD.  These also
       appear in the object_flags field of the bfd_target structure, where
       they indicate the set of flags used by that backend (not all flags
       are meaningful for all object file formats) (FIXME: at the moment,
       the object_flags values have mostly just been copied from backend
       to another, and are not necessarily correct).  */

```

```
#define BFD_NO_FLAGS                0x0

    /* BFD contains relocation entries. */
#define HAS_RELOC                    0x1

    /* BFD is directly executable. */
#define EXEC_P                        0x2

    /* BFD has line number information (basically used for F_LNNO in a
       COFF header). */
#define HAS_LINENO                    0x4

    /* BFD has debugging information. */
#define HAS_DEBUG                      0x08

    /* BFD has symbols. */
#define HAS_SYMS                       0x10

    /* BFD has local symbols (basically used for F_LSYMS in a COFF
       header). */
#define HAS_LOCALS                     0x20

    /* BFD is a dynamic object. */
#define DYNAMIC                        0x40

    /* Text section is write protected (if D_PAGED is not set, this is
       like an a.out NMAGIC file) (the linker sets this by default, but
       clears it for -r or -N). */
#define WP_TEXT                        0x80

    /* BFD is dynamically paged (this is like an a.out ZMAGIC file) (the
       linker sets this by default, but clears it for -r or -n or -N). */
#define D_PAGED                        0x100

    /* BFD is relaxable (this means that bfd_relax_section may be able to
       do something) (sometimes bfd_relax_section can do something even if
       this is not set). */
#define BFD_IS_RELAXABLE               0x200

    /* This may be set before writing out a BFD to request using a
       traditional format. For example, this is used to request that when
       writing out an a.out object the symbols not be hashed to eliminate
       duplicates. */
#define BFD_TRADITIONAL_FORMAT        0x400

    /* This flag indicates that the BFD contents are actually cached
       in memory. If this is set, iostream points to a bfd_in_memory
```

```

    struct. */
#define BFD_IN_MEMORY          0x800

    /* This BFD has been created by the linker and doesn't correspond
       to any input file. */
#define BFD_LINKER_CREATED     0x1000

    /* This may be set before writing out a BFD to request that it
       be written using values for UIDs, GIDs, timestamps, etc. that
       will be consistent from run to run. */
#define BFD_DETERMINISTIC_OUTPUT 0x2000

    /* Compress sections in this BFD. */
#define BFD_COMPRESS          0x4000

    /* Decompress sections in this BFD. */
#define BFD_DECOMPRESS        0x8000

    /* BFD is a dummy, for plugins. */
#define BFD_PLUGIN            0x10000

    /* Compress sections in this BFD with SHF_COMPRESSED from gABI. */
#define BFD_COMPRESS_GABI     0x20000

    /* Convert ELF common symbol type to STT_COMMON or STT_OBJECT in this
       BFD. */
#define BFD_CONVERT_ELF_COMMON 0x40000

    /* Use the ELF STT_COMMON type in this BFD. */
#define BFD_USE_ELF_STT_COMMON 0x80000

    /* Put pathnames into archives (non-POSIX). */
#define BFD_ARCHIVE_FULL_PATH 0x100000

#define BFD_CLOSED_BY_CACHE    0x200000
    /* Compress sections in this BFD with SHF_COMPRESSED zstd. */
#define BFD_COMPRESS_ZSTD     0x400000

    /* Don't generate ELF section header. */
#define BFD_NO_SECTION_HEADER 0x800000

    /* Flags bits which are for BFD use only. */
#define BFD_FLAGS_FOR_BFD_USE_MASK \
    (BFD_IN_MEMORY | BFD_COMPRESS | BFD_DECOMPRESS | BFD_LINKER_CREATED \
     | BFD_PLUGIN | BFD_TRADITIONAL_FORMAT | BFD_DETERMINISTIC_OUTPUT \
     | BFD_COMPRESS_GABI | BFD_CONVERT_ELF_COMMON | BFD_USE_ELF_STT_COMMON \
     | BFD_NO_SECTION_HEADER)

```



```
/* The format which belongs to the BFD. (object, core, etc.) */
ENUM_BITFIELD (bfd_format) format : 3;

/* The direction with which the BFD was opened. */
ENUM_BITFIELD (bfd_direction) direction : 2;

/* Is the file descriptor being cached? That is, can it be closed as
   needed, and re-opened when accessed later? */
unsigned int cacheable : 1;

/* Marks whether there was a default target specified when the
   BFD was opened. This is used to select which matching algorithm
   to use to choose the back end. */
unsigned int target_defaulted : 1;

/* ... and here: ('once' means at least once). */
unsigned int opened_once : 1;

/* Set if we have a locally maintained mtime value, rather than
   getting it from the file each time. */
unsigned int mtime_set : 1;

/* Flag set if symbols from this BFD should not be exported. */
unsigned int no_export : 1;

/* Remember when output has begun, to stop strange things
   from happening. */
unsigned int output_has_begun : 1;

/* Have archive map. */
unsigned int has_armap : 1;

/* Set if this is a thin archive. */
unsigned int is_thin_archive : 1;

/* Set if this archive should not cache element positions. */
unsigned int no_element_cache : 1;

/* Set if only required symbols should be added in the link hash table for
   this object. Used by VMS linkers. */
unsigned int selective_search : 1;

/* Set if this is the linker output BFD. */
unsigned int is_linker_output : 1;

/* Set if this is the linker input BFD. */
```

```
unsigned int is_linker_input : 1;

/* If this is an input for a compiler plug-in library. */
ENUM_BITFIELD (bfd_plugin_format) plugin_format : 2;

/* Set if this is a plugin output file. */
unsigned int lto_output : 1;

/* Set if this is a slim LTO object not loaded with a compiler plugin. */
unsigned int lto_slim_object : 1;

/* Do not attempt to modify this file. Set when detecting errors
   that BFD is not prepared to handle for objcopy/strip. */
unsigned int read_only : 1;

/* Set to dummy BFD created when claimed by a compiler plug-in
   library. */
bfd *plugin_dummy_bfd;

/* The offset of this bfd in the file, typically 0 if it is not
   contained in an archive. */
ufile_ptr origin;

/* The origin in the archive of the proxy entry. This will
   normally be the same as origin, except for thin archives,
   when it will contain the current offset of the proxy in the
   thin archive rather than the offset of the bfd in its actual
   container. */
ufile_ptr proxy_origin;

/* A hash table for section names. */
struct bfd_hash_table section_htab;

/* Pointer to linked list of sections. */
struct bfd_section *sections;

/* The last section on the section list. */
struct bfd_section *section_last;

/* The number of sections. */
unsigned int section_count;

/* The archive plugin file descriptor. */
int archive_plugin_fd;

/* The number of opens on the archive plugin file descriptor. */
unsigned int archive_plugin_fd_open_count;
```

```

/* A field used by _bfd_generic_link_add_archive_symbols. This will
   be used only for archive elements. */
int archive_pass;

/* The total size of memory from bfd_alloc. */
bfd_size_type alloc_size;

/* Stuff only useful for object files:
   The start address. */
bfd_vma start_address;

/* Symbol table for output BFD (with symcount entries).
   Also used by the linker to cache input BFD symbols. */
struct bfd_symbol **outsymbols;

/* Used for input and output. */
unsigned int symcount;

/* Used for slurped dynamic symbol tables. */
unsigned int dynsymcount;

/* Pointer to structure which contains architecture information. */
const struct bfd_arch_info *arch_info;

/* Cached length of file for bfd_get_size. 0 until bfd_get_size is
   called, 1 if stat returns an error or the file size is too large to
   return in ufile_ptr. Both 0 and 1 should be treated as "unknown". */
ufile_ptr size;

/* Stuff only useful for archives. */
void *arelt_data;
struct bfd *my_archive; /* The containing archive BFD. */
struct bfd *archive_next; /* The next BFD in the archive. */
struct bfd *archive_head; /* The first BFD in the archive. */
struct bfd *nested_archives; /* List of nested archive in a flattened
                               thin archive. */

union {
  /* For input BFDs, a chain of BFDs involved in a link. */
  struct bfd *next;
  /* For output BFD, the linker hash table. */
  struct bfd_link_hash_table *hash;
} link;

/* Used by the back end to hold private data. */
union

```

```

{
  struct aout_data_struct *aout_data;
  struct artdata *aout_ar_data;
  struct coff_tdata *coff_obj_data;
  struct pe_tdata *pe_obj_data;
  struct xcoff_tdata *xcoff_obj_data;
  struct ecoff_tdata *ecoff_obj_data;
  struct srec_data_struct *srec_data;
  struct verilog_data_struct *verilog_data;
  struct ihex_data_struct *ihex_data;
  struct tekhex_data_struct *tekhex_data;
  struct elf_obj_tdata *elf_obj_data;
  struct mmo_data_struct *mmo_data;
  struct trad_core_struct *trad_core_data;
  struct som_data_struct *som_data;
  struct hpux_core_struct *hpux_core_data;
  struct hppabsd_core_struct *hppabsd_core_data;
  struct sgi_core_struct *sgi_core_data;
  struct lynx_core_struct *lynx_core_data;
  struct osf_core_struct *osf_core_data;
  struct cisco_core_struct *cisco_core_data;
  struct netbsd_core_struct *netbsd_core_data;
  struct mach_o_data_struct *mach_o_data;
  struct mach_o_fat_data_struct *mach_o_fat_data;
  struct plugin_data_struct *plugin_data;
  struct bfd_pef_data_struct *pef_data;
  struct bfd_pef_xlib_data_struct *pef_xlib_data;
  struct bfd_sym_data_struct *sym_data;
  void *any;
}
tdata;

/* Used by the application to hold private data. */
void *usrdata;

/* Where all the allocated stuff under this BFD goes. This is a
   struct objalloc *, but we use void * to avoid requiring the inclusion
   of objalloc.h. */
void *memory;

/* For input BFDs, the build ID, if the object has one. */
const struct bfd_build_id *build_id;
};

```

2.2 Error reporting

Most BFD functions return nonzero on success (check their individual documentation for precise semantics). On an error, they call `bfd_set_error` to set an error condition that callers can check by calling `bfd_get_error`. If that returns `bfd_error_system_call`, then check `errno`.

The easiest way to report a BFD error to the user is to use `bfd_perror`.

2.2.1 Type `bfd_error_type`

The values returned by `bfd_get_error` are defined by the enumerated type `bfd_error_type`.

```
typedef enum bfd_error
{
    bfd_error_no_error = 0,
    bfd_error_system_call,
    bfd_error_invalid_target,
    bfd_error_wrong_format,
    bfd_error_wrong_object_format,
    bfd_error_invalid_operation,
    bfd_error_no_memory,
    bfd_error_no_symbols,
    bfd_error_no_armap,
    bfd_error_no_more_archived_files,
    bfd_error_malformed_archive,
    bfd_error_missing_dso,
    bfd_error_file_not_recognized,
    bfd_error_file_ambiguously_recognized,
    bfd_error_no_contents,
    bfd_error_nonrepresentable_section,
    bfd_error_no_debug_section,
    bfd_error_bad_value,
    bfd_error_file_truncated,
    bfd_error_file_too_big,
    bfd_error_sorry,
    bfd_error_on_input,
    bfd_error_invalid_error_code
}
bfd_error_type;
```

2.2.1.1 `bfd_get_error`

```
bfd_error_type bfd_get_error (void);
```

[Function]

Return the current BFD error condition.

2.2.1.2 `bfd_set_error`

```
void bfd_set_error (bfd_error_type error_tag);
```

[Function]

Set the BFD error condition to be `error_tag`.

error_tag must not be `bfd_error_on_input`. Use `bfd_set_input_error` for input errors instead.

2.2.1.3 `bfd_set_input_error`

```
void bfd_set_input_error (bfd *input, bfd_error_type      [Function]
                        error_tag);
```

Set the BFD error condition to be `bfd_error_on_input`. *input* is the input bfd where the error occurred, and *error_tag* the `bfd_error_type` error.

2.2.1.4 `bfd_errmsg`

```
const char *bfd_errmsg (bfd_error_type error_tag);          [Function]
Return a string describing the error error_tag, or the system error if error_tag is
bfd_error_system_call.
```

2.2.1.5 `bfd_perror`

```
void bfd_perror (const char *message);                      [Function]
Print to the standard error stream a string describing the last BFD error that occurred,
or the last system error if the last BFD error was a system call failure. If message is
non-NULL and non-empty, the error string printed is preceded by message, a colon,
and a space. It is followed by a newline.
```

2.2.1.6 `bfd_asprintf`

```
char *bfd_asprintf (const char *fmt, ...);                 [Function]
Primarily for error reporting, this function is like liberty's xasprintf except that it
can return NULL on no memory and the returned string should not be freed. Uses a
single malloc'd buffer managed by libbfd, _bfd_error_buf. Be aware that a call to this
function frees the result of any previous call. bfd_errmsg (bfd_error_on_input) also
calls this function.
```

2.2.2 BFD error handler

Some BFD functions want to print messages describing the problem. They call a BFD error handler function. This function may be overridden by the program.

The BFD error handler acts like `vprintf`.

```
typedef void (*bfd_error_handler_type) (const char *, va_list);
```

2.2.2.1 `_bfd_error_handler`

```
void _bfd_error_handler (const char *fmt, ...)             [Function]
ATTRIBUTE_PRINTF_1;
```

This is the default routine to handle BFD error messages. Like `fprintf (stderr, ...)`, but also handles some extra format specifiers.

`%pA` section name from section. For group components, prints group name too. `%pB` file name from bfd. For archive components, prints archive too.

Beware: Only supports a maximum of 9 format arguments.

2.2.2.2 bfd_set_error_handler

```
bfd_error_handler_type bfd_set_error_handler      [Function]
    (bfd_error_handler_type);
```

Set the BFD error handler function. Returns the previous function.

2.2.2.3 _bfd_set_error_handler_caching

```
bfd_error_handler_type _bfd_set_error_handler_caching (bfd      [Function]
    *);
```

Set the BFD error handler function to one that stores messages to the per_xvec_warn array. Returns the previous function.

2.2.2.4 bfd_set_error_program_name

```
void bfd_set_error_program_name (const char *);      [Function]
```

Set the program name to use when printing a BFD error. This is printed before the error message followed by a colon and space. The string must not be changed after it is passed to this function.

2.2.2.5 _bfd_get_error_program_name

```
const char *_bfd_get_error_program_name (void);      [Function]
```

Get the program name used when printing a BFD error.

2.2.3 BFD assert handler

If BFD finds an internal inconsistency, the bfd assert handler is called with information on the BFD version, BFD source file and line. If this happens, most programs linked against BFD are expected to want to exit with an error, or mark the current BFD operation as failed, so it is recommended to override the default handler, which just calls _bfd_error_handler and continues.

```
typedef void (*bfd_assert_handler_type) (const char *bfd_formatmsg,
                                         const char *bfd_version,
                                         const char *bfd_file,
                                         int bfd_line);
```

2.2.3.1 bfd_set_assert_handler

```
bfd_assert_handler_type bfd_set_assert_handler      [Function]
    (bfd_assert_handler_type);
```

Set the BFD assert handler function. Returns the previous function.

2.2.3.2 bfd_init

```
unsigned int bfd_init (void);      [Function]
```

This routine must be called before any other BFD function to initialize magical internal data structures. Returns a magic number, which may be used to check that the bfd library is configured as expected by users.

```
/* Value returned by bfd_init. */
```

```
#define BFD_INIT_MAGIC (sizeof (struct bfd_section))
```

2.3 Miscellaneous

2.3.1 Miscellaneous functions

2.3.1.1 bfd_get_reloc_upper_bound

```
long bfd_get_reloc_upper_bound (bfd *abfd, asection *sect); [Function]
```

Return the number of bytes required to store the relocation information associated with section *sect* attached to bfd *abfd*. If an error occurs, return -1.

2.3.1.2 bfd_canonicalize_reloc

```
long bfd_canonicalize_reloc (bfd *abfd, asection *sec,          [Function]
                             arelent **loc, asymbol **syms);
```

Call the back end associated with the open BFD *abfd* and translate the external form of the relocation information attached to *sec* into the internal canonical form. Place the table into memory at *loc*, which has been preallocated, usually by a call to `bfd_get_reloc_upper_bound`. Returns the number of relocs, or -1 on error. The *syms* table is also needed for horrible internal magic reasons.

2.3.1.3 bfd_set_reloc

```
void bfd_set_reloc (bfd *abfd, asection *sec, arelent          [Function]
                   **rel, unsigned int count);
```

Set the relocation pointer and count within section *sec* to the values *rel* and *count*. The argument *abfd* is ignored.

```
#define bfd_set_reloc(abfd, asect, location, count) \
    BFD_SEND (abfd, _bfd_set_reloc, (abfd, asect, location, count))■
```

2.3.1.4 bfd_set_file_flags

```
bool bfd_set_file_flags (bfd *abfd, flagword flags);          [Function]
```

Set the flag word in the BFD *abfd* to the value *flags*. Possible errors are:

- `bfd_error_wrong_format` - The target bfd was not of object format.
- `bfd_error_invalid_operation` - The target bfd was open for reading.
- `bfd_error_invalid_operation` - The flag word contained a bit which was not applicable to the type of file. E.g., an attempt was made to set the `D_PAGED` bit on a BFD format which does not support demand paging.

2.3.1.5 bfd_get_arch_size

```
int bfd_get_arch_size (bfd *abfd);                             [Function]
```

Returns the normalized architecture address size, in bits, as determined by the object file's format. By normalized, we mean either 32 or 64. For ELF, this information

is included in the header. Use `bfd_arch_bits_per_address` for number of bits in the architecture address.

Returns the arch size in bits if known, -1 otherwise.

2.3.1.6 `bfd_get_sign_extend_vma`

`int bfd_get_sign_extend_vma (bfd *abfd);` [Function]

Indicates if the target architecture "naturally" sign extends an address. Some architectures implicitly sign extend address values when they are converted to types larger than the size of an address. For instance, `bfd_get_start_address()` will return an address sign extended to fill a `bfd_vma` when this is the case.

Returns 1 if the target architecture is known to sign extend addresses, 0 if the target architecture is known to not sign extend addresses, and -1 otherwise.

2.3.1.7 `bfd_set_start_address`

`bool bfd_set_start_address (bfd *abfd, bfd_vma vma);` [Function]

Make *vma* the entry point of output BFD *abfd*.

Returns TRUE on success, FALSE otherwise.

2.3.1.8 `bfd_get_gp_size`

`unsigned int bfd_get_gp_size (bfd *abfd);` [Function]

Return the maximum size of objects to be optimized using the GP register under MIPS ECOFF. This is typically set by the `-G` argument to the compiler, assembler or linker.

2.3.1.9 `bfd_set_gp_size`

`void bfd_set_gp_size (bfd *abfd, unsigned int i);` [Function]

Set the maximum size of objects to be optimized using the GP register under ECOFF or MIPS ELF. This is typically set by the `-G` argument to the compiler, assembler or linker.

2.3.1.10 `bfd_set_gp_value`

`void bfd_set_gp_value (bfd *abfd, bfd_vma v);` [Function]

Allow external access to the function to set the GP value. This is specifically added for gdb-compile support.

2.3.1.11 `bfd_scan_vma`

`bfd_vma bfd_scan_vma (const char *string, const char **end, int base);` [Function]

Convert, like `strtoul`, a numerical expression *string* into a `bfd_vma` integer, and return that integer. (Though without as many bells and whistles as `strtoul`.) The expression is assumed to be unsigned (i.e., positive). If given a *base*, it is used as the base for conversion. A base of 0 causes the function to interpret the string in hex if a

leading "0x" or "0X" is found, otherwise in octal if a leading zero is found, otherwise in decimal.

If the value would overflow, the maximum `bfd_vma` value is returned.

2.3.1.12 `bfd_copy_private_header_data`

`bool bfd_copy_private_header_data (bfd *ibfd, bfd *obfd);` [Function]
Copy private BFD header information from the BFD *ibfd* to the the BFD *obfd*. This copies information that may require sections to exist, but does not require symbol tables. Return `true` on success, `false` on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *obfd*.

```
#define bfd_copy_private_header_data(ibfd, obfd) \
    BFD_SEND (obfd, _bfd_copy_private_header_data, \
              (ibfd, obfd))
```

2.3.1.13 `bfd_copy_private_bfd_data`

`bool bfd_copy_private_bfd_data (bfd *ibfd, bfd *obfd);` [Function]
Copy private BFD information from the BFD *ibfd* to the the BFD *obfd*. Return `TRUE` on success, `FALSE` on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *obfd*.

```
#define bfd_copy_private_bfd_data(ibfd, obfd) \
    BFD_SEND (obfd, _bfd_copy_private_bfd_data, \
              (ibfd, obfd))
```

2.3.1.14 `bfd_set_private_flags`

`bool bfd_set_private_flags (bfd *abfd, flagword flags);` [Function]
Set private BFD flag information in the BFD *abfd*. Return `TRUE` on success, `FALSE` on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *obfd*.

```
#define bfd_set_private_flags(abfd, flags) \
    BFD_SEND (abfd, _bfd_set_private_flags, (abfd, flags))
```

2.3.1.15 Other functions

The following functions exist but have not yet been documented.

```
#define bfd_sizeof_headers(abfd, info) \
    BFD_SEND (abfd, _bfd_sizeof_headers, (abfd, info))
```

```
#define bfd_find_nearest_line(abfd, sec, syms, off, file, func, line) \
    BFD_SEND (abfd, _bfd_find_nearest_line, \
              (abfd, syms, sec, off, file, func, line, NULL))
```

```

#define bfd_find_nearest_line_with_alt(abfd, alt_filename, sec, syms, off, \
                                     file, func, line, disc) \
    BFD_SEND (abfd, _bfd_find_nearest_line_with_alt, \
              (abfd, alt_filename, syms, sec, off, file, func, line, disc))

#define bfd_find_nearest_line_discriminator(abfd, sec, syms, off, file, func, \
                                           line, disc) \
    BFD_SEND (abfd, _bfd_find_nearest_line, \
              (abfd, syms, sec, off, file, func, line, disc))

#define bfd_find_line(abfd, syms, sym, file, line) \
    BFD_SEND (abfd, _bfd_find_line, \
              (abfd, syms, sym, file, line))

#define bfd_find_inliner_info(abfd, file, func, line) \
    BFD_SEND (abfd, _bfd_find_inliner_info, \
              (abfd, file, func, line))

#define bfd_debug_info_start(abfd) \
    BFD_SEND (abfd, _bfd_debug_info_start, (abfd))

#define bfd_debug_info_end(abfd) \
    BFD_SEND (abfd, _bfd_debug_info_end, (abfd))

#define bfd_debug_info_accumulate(abfd, section) \
    BFD_SEND (abfd, _bfd_debug_info_accumulate, (abfd, section))

#define bfd_stat_arch_elt(abfd, stat) \
    BFD_SEND (abfd->my_archive ? abfd->my_archive : abfd, \
              _bfd_stat_arch_elt, (abfd, stat))

#define bfd_update_armap_timestamp(abfd) \
    BFD_SEND (abfd, _bfd_update_armap_timestamp, (abfd))

#define bfd_set_arch_mach(abfd, arch, mach)\
    BFD_SEND ( abfd, _bfd_set_arch_mach, (abfd, arch, mach))

#define bfd_relax_section(abfd, section, link_info, again) \
    BFD_SEND (abfd, _bfd_relax_section, (abfd, section, link_info, again))

#define bfd_gc_sections(abfd, link_info) \
    BFD_SEND (abfd, _bfd_gc_sections, (abfd, link_info))

#define bfd_lookup_section_flags(link_info, flag_info, section) \
    BFD_SEND (abfd, _bfd_lookup_section_flags, (link_info, flag_info, section))

#define bfd_merge_sections(abfd, link_info) \

```

```

        BFD_SEND (abfd, _bfd_merge_sections, (abfd, link_info))

#define bfd_is_group_section(abfd, sec) \
        BFD_SEND (abfd, _bfd_is_group_section, (abfd, sec))

#define bfd_group_name(abfd, sec) \
        BFD_SEND (abfd, _bfd_group_name, (abfd, sec))

#define bfd_discard_group(abfd, sec) \
        BFD_SEND (abfd, _bfd_discard_group, (abfd, sec))

#define bfd_link_hash_table_create(abfd) \
        BFD_SEND (abfd, _bfd_link_hash_table_create, (abfd))

#define bfd_link_add_symbols(abfd, info) \
        BFD_SEND (abfd, _bfd_link_add_symbols, (abfd, info))

#define bfd_link_just_syms(abfd, sec, info) \
        BFD_SEND (abfd, _bfd_link_just_syms, (sec, info))

#define bfd_final_link(abfd, info) \
        BFD_SEND (abfd, _bfd_final_link, (abfd, info))

#define bfd_free_cached_info(abfd) \
        BFD_SEND (abfd, _bfd_free_cached_info, (abfd))

#define bfd_get_dynamic_symtab_upper_bound(abfd) \
        BFD_SEND (abfd, _bfd_get_dynamic_symtab_upper_bound, (abfd))

#define bfd_print_private_bfd_data(abfd, file)\
        BFD_SEND (abfd, _bfd_print_private_bfd_data, (abfd, file))

#define bfd_canonicalize_dynamic_symtab(abfd, asymbols) \
        BFD_SEND (abfd, _bfd_canonicalize_dynamic_symtab, (abfd, asymbols))■

#define bfd_get_synthetic_symtab(abfd, count, syms, dyncount, dynsyms, ret) \
        BFD_SEND (abfd, _bfd_get_synthetic_symtab, (abfd, count, syms, \
        dyncount, dynsyms, ret))■

#define bfd_get_dynamic_reloc_upper_bound(abfd) \
        BFD_SEND (abfd, _bfd_get_dynamic_reloc_upper_bound, (abfd))

#define bfd_canonicalize_dynamic_reloc(abfd, arels, asyms) \
        BFD_SEND (abfd, _bfd_canonicalize_dynamic_reloc, (abfd, arels, asyms))■

```

2.3.1.16 `bfd_get_relocated_section_contents`

```
bfd_byte *bfd_get_relocated_section_contents (bfd *, [Function]
      struct bfd_link_info *, struct bfd_link_order *, bfd_byte *,
      bool, asymbol **);
```

Read and relocate the indirect link_order section, into DATA (if non-NULL) or to a malloc'd buffer. Return the buffer, or NULL on errors.

2.3.1.17 `bfd_record_phdr`

```
bool bfd_record_phdr (bfd *, unsigned long, bool, [Function]
      flagword, bool, bfd_vma, bool, bool, unsigned int, struct
      bfd_section **);
```

Record information about an ELF program header.

2.3.1.18 `bfd_sprintf_vma`

```
void bfd_sprintf_vma (bfd *, char *, bfd_vma); void [Function]
      bfd_fprintf_vma (bfd *, void *, bfd_vma);
```

`bfd_sprintf_vma` and `bfd_fprintf_vma` display an address in the target's address size.

2.3.1.19 `bfd_alt_mach_code`

```
bool bfd_alt_mach_code (bfd *abfd, int alternative); [Function]
```

When more than one machine code number is available for the same machine type, this function can be used to switch between the preferred one (`alternative == 0`) and any others. Currently, only ELF supports this feature, with up to two alternate machine codes.

2.3.1.20 `bfd_emul_get_maxpagesize`

```
bfd_vma bfd_emul_get_maxpagesize (const char *); [Function]
```

Returns the maximum page size, in bytes, as determined by emulation.

2.3.1.21 `bfd_emul_get_commonpagesize`

```
bfd_vma bfd_emul_get_commonpagesize (const char *); [Function]
```

Returns the common page size, in bytes, as determined by emulation.

2.3.1.22 `bfd_demangle`

```
char *bfd_demangle (bfd *, const char *, int); [Function]
```

Wrapper around `cplus_demangle`. Strips leading underscores and other such chars that would otherwise confuse the demangler. If passed a g++ v3 ABI mangled name, returns a buffer allocated with malloc holding the demangled name. Returns NULL otherwise and on memory alloc failure.

2.3.1.23 struct bfd_iovec

The struct `bfd_iovec` contains the internal file I/O class. Each BFD has an instance of this class and all file I/O is routed through it (it is assumed that the instance implements all methods listed below).

```

struct bfd_iovec
{
    /* To avoid problems with macros, a "b" rather than "f"
       prefix is prepended to each method name. */
    /* Attempt to read/write NBYTES on ABFD's IOSTREAM storing/fetching
       bytes starting at PTR. Return the number of bytes actually
       transferred (a read past end-of-file returns less than NBYTES),
       or -1 (setting bfd_error) if an error occurs. */
    file_ptr (*bread) (struct bfd *abfd, void *ptr, file_ptr nbytes);
    file_ptr (*bwrite) (struct bfd *abfd, const void *ptr,
                       file_ptr nbytes);
    /* Return the current IOSTREAM file offset, or -1 (setting bfd_error
       if an error occurs. */
    file_ptr (*btell) (struct bfd *abfd);
    /* For the following, on successful completion a value of 0 is returned.
       Otherwise, a value of -1 is returned (and bfd_error is set). */
    int (*bseek) (struct bfd *abfd, file_ptr offset, int whence);
    int (*bclose) (struct bfd *abfd);
    int (*bflush) (struct bfd *abfd);
    int (*bstat) (struct bfd *abfd, struct stat *sb);
    /* Mmap a part of the files. ADDR, LEN, PROT, FLAGS and OFFSET are the usual
       mmap parameter, except that LEN and OFFSET do not need to be page
       aligned. Returns (void *)-1 on failure, mmapped address on success.
       Also write in MAP_ADDR the address of the page aligned buffer and in
       MAP_LEN the size mapped (a page multiple). Use unmap with MAP_ADDR and
       MAP_LEN to unmap. */
    void *(*bmmmap) (struct bfd *abfd, void *addr, bfd_size_type len,
                   int prot, int flags, file_ptr offset,
                   void **map_addr, bfd_size_type *map_len);
};
extern const struct bfd_iovec _bfd_memory_iovec;

```

2.3.1.24 bfd_bread

```

bfd_size_type bfd_bread (void *, bfd_size_type, bfd *);          [Function]
    Attempt to read SIZE bytes from ABFD's iostream to PTR. Return the amount read.

```

2.3.1.25 `bfd_bwrite`

`bfd_size_type` `bfd_bwrite` (`const void *`, `bfd_size_type`, `bfd` [Function]
`*`);

Attempt to write `SIZE` bytes to ABFD's iostream from `PTR`. Return the amount written.

2.3.1.26 `bfd_tell`

`file_ptr` `bfd_tell` (`bfd *`); [Function]

Return ABFD's iostream file position.

2.3.1.27 `bfd_flush`

`int` `bfd_flush` (`bfd *`); [Function]

Flush ABFD's iostream pending IO.

2.3.1.28 `bfd_stat`

`int` `bfd_stat` (`bfd *`, `struct stat *`); [Function]

Call `fstat` on ABFD's iostream. Return 0 on success, and a negative value on failure.

2.3.1.29 `bfd_seek`

`int` `bfd_seek` (`bfd *`, `file_ptr`, `int`); [Function]

Call `fseek` on ABFD's iostream. Return 0 on success, and a negative value on failure.

2.3.1.30 `bfd_get_mtime`

`long` `bfd_get_mtime` (`bfd *abfd`); [Function]

Return the file modification time (as read from the file system, or from the archive header for archive members).

2.3.1.31 `bfd_get_size`

`ufile_ptr` `bfd_get_size` (`bfd *abfd`); [Function]

Return the file size (as read from file system) for the file associated with BFD *abfd*.

The initial motivation for, and use of, this routine is not so we can get the exact size of the object the BFD applies to, since that might not be generally possible (archive members for example). It would be ideal if someone could eventually modify it so that such results were guaranteed.

Instead, we want to ask questions like "is this NNN byte sized object I'm about to try read from file offset YYY reasonable?" As an example of where we might do this, some object formats use string tables for which the first `sizeof (long)` bytes of the table contain the size of the table itself, including the size bytes. If an application tries to read what it thinks is one of these string tables, without some way to validate the size, and for some reason the size is wrong (byte swapping error, wrong location for the string table, etc.), the only clue is likely to be a read error when it tries to read the table, or a "virtual memory exhausted" error when it tries to allocate 15 bazillion

bytes of space for the 15 bazillion byte table it is about to read. This function at least allows us to answer the question, "is the size reasonable?".

A return value of zero indicates the file size is unknown.

2.3.1.32 `bfd_get_file_size`

`ufile_ptr bfd_get_file_size (bfd *abfd);` [Function]

Return the file size (as read from file system) for the file associated with BFD *abfd*. It supports both normal files and archive elements.

2.3.1.33 `bfd_mmap`

`void *bfd_mmap (bfd *abfd, void *addr, bfd_size_type len, [Function]
int prot, int flags, file_ptr offset, void **map_addr,
bfd_size_type *map_len);`

Return `mmap()`ed region of the file, if possible and implemented. `LEN` and `OFFSET` do not need to be page aligned. The page aligned address and length are written to `MAP_ADDR` and `MAP_LEN`.

2.4 Memory Usage

BFD keeps all of its internal structures in obstacks. There is one obstack per open BFD file, into which the current state is stored. When a BFD is closed, the obstack is deleted, and so everything which has been allocated by BFD for the closing file is thrown away.

BFD does not free anything created by an application, but pointers into `bfd` structures become invalid on a `bfd_close`; for example, after a `bfd_close` the vector passed to `bfd_canonicalize_symtab` is still around, since it has been allocated by the application, but the data that it pointed to are lost.

The general rule is to not close a BFD until all operations dependent upon data from the BFD have been completed, or all the data from within the file has been copied. To help with the management of memory, there is a function (`bfd_alloc_size`) which returns the number of bytes in obstacks associated with the supplied BFD. This could be used to select the greediest open BFD, close it to reclaim the memory, perform some operation and reopen the BFD again, to get a fresh copy of the data structures.

2.5 Sections

The raw data contained within a BFD is maintained through the section abstraction. A single BFD may have any number of sections. It keeps hold of them by pointing to the first; each one points to the next in the list.

Sections are supported in BFD in `section.c`.

2.5.1 Section input

When a BFD is opened for reading, the section structures are created and attached to the BFD.

Each section has a name which describes the section in the outside world—for example, `a.out` would contain at least three sections, called `.text`, `.data` and `.bss`.

relocations to see if any are attached to data which can be shrunk, if so it does it on a link_order by link_order basis.

2.5.4 typedef asection

Here is the section structure:

```
typedef struct bfd_section
{
  /* The name of the section; the name isn't a copy, the pointer is
     the same as that passed to bfd_make_section.  */
  const char *name;

  /* The next section in the list belonging to the BFD, or NULL.  */
  struct bfd_section *next;

  /* The previous section in the list belonging to the BFD, or NULL.  */
  struct bfd_section *prev;

  /* A unique sequence number.  */
  unsigned int id;

  /* A unique section number which can be used by assembler to
     distinguish different sections with the same section name.  */
  unsigned int section_id;

  /* Which section in the bfd; 0..n-1 as sections are created in a bfd.  */
  unsigned int index;

  /* The field flags contains attributes of the section. Some
     flags are read in from the object file, and some are
     synthesized from other information.  */
  flagword flags;

#define SEC_NO_FLAGS                0x0

  /* Tells the OS to allocate space for this section when loading.
     This is clear for a section containing debug information only.  */
#define SEC_ALLOC                   0x1

  /* Tells the OS to load the section from the file when loading.
     This is clear for a .bss section.  */
#define SEC_LOAD                     0x2

  /* The section contains data still to be relocated, so there is
     some relocation information too.  */
#define SEC_RELOC                   0x4

```

```

    /* A signal to the OS that the section contains read only data. */
#define SEC_READONLY                0x8

    /* The section contains code only. */
#define SEC_CODE                    0x10

    /* The section contains data only. */
#define SEC_DATA                    0x20

    /* The section will reside in ROM. */
#define SEC_ROM                    0x40

    /* The section contains constructor information. This section
       type is used by the linker to create lists of constructors and
       destructors used by g++. When a back end sees a symbol
       which should be used in a constructor list, it creates a new
       section for the type of name (e.g., __CTOR_LIST__), attaches
       the symbol to it, and builds a relocation. To build the lists
       of constructors, all the linker has to do is catenate all the
       sections called __CTOR_LIST__ and relocate the data
       contained within - exactly the operations it would perform on
       standard data. */
#define SEC_CONSTRUCTOR            0x80

    /* The section has contents - a data section could be
       SEC_ALLOC | SEC_HAS_CONTENTS; a debug section could be
       SEC_HAS_CONTENTS */
#define SEC_HAS_CONTENTS          0x100

    /* An instruction to the linker to not output the section
       even if it has information which would normally be written. */
#define SEC_NEVER_LOAD            0x200

    /* The section contains thread local data. */
#define SEC_THREAD_LOCAL          0x400

    /* The section's size is fixed. Generic linker code will not
       recalculate it and it is up to whoever has set this flag to
       get the size right. */
#define SEC_FIXED_SIZE            0x800

    /* The section contains common symbols (symbols may be defined
       multiple times, the value of a symbol is the amount of
       space it requires, and the largest symbol value is the one
       used). Most targets have exactly one of these (which we
       translate to bfd_com_section_ptr), but ECOFF has two. */
#define SEC_IS_COMMON            0x1000

```

```

/* The section contains only debugging information. For
   example, this is set for ELF .debug and .stab sections.
   strip tests this flag to see if a section can be
   discarded. */
#define SEC_DEBUGGING                0x2000

/* The contents of this section are held in memory pointed to
   by the contents field. This is checked by bfd_get_section_contents,
   and the data is retrieved from memory if appropriate. */
#define SEC_IN_MEMORY                0x4000

/* The contents of this section are to be excluded by the
   linker for executable and shared objects unless those
   objects are to be further relocated. */
#define SEC_EXCLUDE                  0x8000

/* The contents of this section are to be sorted based on the sum of
   the symbol and addend values specified by the associated relocation
   entries. Entries without associated relocation entries will be
   appended to the end of the section in an unspecified order. */
#define SEC_SORT_ENTRIES              0x10000

/* When linking, duplicate sections of the same name should be
   discarded, rather than being combined into a single section as
   is usually done. This is similar to how common symbols are
   handled. See SEC_LINK_DUPLICATES below. */
#define SEC_LINK_ONCE                 0x20000

/* If SEC_LINK_ONCE is set, this bitfield describes how the linker
   should handle duplicate sections. */
#define SEC_LINK_DUPLICATES           0xc0000

/* This value for SEC_LINK_DUPLICATES means that duplicate
   sections with the same name should simply be discarded. */
#define SEC_LINK_DUPLICATES_DISCARD   0x0

/* This value for SEC_LINK_DUPLICATES means that the linker
   should warn if there are any duplicate sections, although
   it should still only link one copy. */
#define SEC_LINK_DUPLICATES_ONE_ONLY  0x40000

/* This value for SEC_LINK_DUPLICATES means that the linker
   should warn if any duplicate sections are a different size. */
#define SEC_LINK_DUPLICATES_SAME_SIZE 0x80000

/* This value for SEC_LINK_DUPLICATES means that the linker

```

```
    should warn if any duplicate sections contain different
    contents. */
#define SEC_LINK_DUPLICATES_SAME_CONTENTS \
    (SEC_LINK_DUPLICATES_ONE_ONLY | SEC_LINK_DUPLICATES_SAME_SIZE)

/* This section was created by the linker as part of dynamic
   relocation or other arcane processing. It is skipped when
   going through the first-pass output, trusting that someone
   else up the line will take care of it later. */
#define SEC_LINKER_CREATED          0x100000

/* This section contains a section ID to distinguish different
   sections with the same section name. */
#define SEC_ASSEMBLER_SECTION_ID   0x100000

/* This section should not be subject to garbage collection.
   Also set to inform the linker that this section should not be
   listed in the link map as discarded. */
#define SEC_KEEP                    0x200000

/* This section contains "short" data, and should be placed
   "near" the GP. */
#define SEC_SMALL_DATA              0x400000

/* Attempt to merge identical entities in the section.
   Entity size is given in the entsize field. */
#define SEC_MERGE                   0x800000

/* If given with SEC_MERGE, entities to merge are zero terminated
   strings where entsize specifies character size instead of fixed
   size entries. */
#define SEC_STRINGS                  0x1000000

/* This section contains data about section groups. */
#define SEC_GROUP                    0x2000000

/* The section is a COFF shared library section. This flag is
   only for the linker. If this type of section appears in
   the input file, the linker must copy it to the output file
   without changing the vma or size. FIXME: Although this
   was originally intended to be general, it really is COFF
   specific (and the flag was renamed to indicate this). It
   might be cleaner to have some more general mechanism to
   allow the back end to control what the linker does with
   sections. */
#define SEC_COFF_SHARED_LIBRARY     0x4000000
```

```

    /* This input section should be copied to output in reverse order
       as an array of pointers. This is for ELF linker internal use
       only. */
#define SEC_ELF_REVERSE_COPY      0x4000000

    /* This section contains data which may be shared with other
       executables or shared objects. This is for COFF only. */
#define SEC_COFF_SHARED          0x8000000

    /* Indicate that section has the purecode flag set. */
#define SEC_ELF_PURECODE         0x8000000

    /* When a section with this flag is being linked, then if the size of
       the input section is less than a page, it should not cross a page
       boundary. If the size of the input section is one page or more,
       it should be aligned on a page boundary. This is for TI
       TMS320C54X only. */
#define SEC_TIC54X_BLOCK        0x10000000

    /* Conditionally link this section; do not link if there are no
       references found to any symbol in the section. This is for TI
       TMS320C54X only. */
#define SEC_TIC54X_CLINK        0x20000000

    /* This section contains vliw code. This is for Toshiba MeP only. */
#define SEC_MEP_VLIW            0x20000000

    /* All symbols, sizes and relocations in this section are octets
       instead of bytes. Required for DWARF debug sections as DWARF
       information is organized in octets, not bytes. */
#define SEC_ELF_OCTETS          0x40000000

    /* Indicate that section has the no read flag set. This happens
       when memory read flag isn't set. */
#define SEC_COFF_NOREAD         0x40000000

    /* End of section flags. */

    /* Some internal packed boolean fields. */

    /* See the vma field. */
    unsigned int user_set_vma : 1;

    /* A mark flag used by some of the linker backends. */
    unsigned int linker_mark : 1;

    /* Another mark flag used by some of the linker backends. Set for

```

```
    output sections that have an input section. */
    unsigned int linker_has_input : 1;

    /* Mark flag used by some linker backends for garbage collection. */
    unsigned int gc_mark : 1;

    /* Section compression status. */
    unsigned int compress_status : 2;
#define COMPRESS_SECTION_NONE    0
#define COMPRESS_SECTION_DONE    1
#define DECOMPRESS_SECTION_ZLIB  2
#define DECOMPRESS_SECTION_ZSTD  3

    /* The following flags are used by the ELF linker. */

    /* Mark sections which have been allocated to segments. */
    unsigned int segment_mark : 1;

    /* Type of sec_info information. */
    unsigned int sec_info_type:3;
#define SEC_INFO_TYPE_NONE        0
#define SEC_INFO_TYPE_STABS       1
#define SEC_INFO_TYPE_MERGE       2
#define SEC_INFO_TYPE_EH_FRAME    3
#define SEC_INFO_TYPE_JUST_SYMS   4
#define SEC_INFO_TYPE_TARGET      5
#define SEC_INFO_TYPE_EH_FRAME_ENTRY 6
#define SEC_INFO_TYPE_SFRAME      7

    /* Nonzero if this section uses RELA relocations, rather than REL. */
    unsigned int use_rela_p:1;

    /* Bits used by various backends.  The generic code doesn't touch
       these fields. */

    unsigned int sec_flg0:1;
    unsigned int sec_flg1:1;
    unsigned int sec_flg2:1;
    unsigned int sec_flg3:1;
    unsigned int sec_flg4:1;
    unsigned int sec_flg5:1;

    /* End of internal packed boolean fields. */

    /* The virtual memory address of the section - where it will be
       at run time.  The symbols are relocated against this.  The
       user_set_vma flag is maintained by bfd; if it's not set, the
```

```
        backend can assign addresses (for example, in a.out, where
        the default address for .data is dependent on the specific
        target and various flags). */
bfd_vma vma;

/* The load address of the section - where it would be in a
   rom image; really only used for writing section header
   information. */
bfd_vma lma;

/* The size of the section in *octets*, as it will be output.
   Contains a value even if the section has no contents (e.g., the
   size of .bss). */
bfd_size_type size;

/* For input sections, the original size on disk of the section, in
   octets. This field should be set for any section whose size is
   changed by linker relaxation. It is required for sections where
   the linker relaxation scheme doesn't cache altered section and
   reloc contents (stabs, eh_frame, SEC_MERGE, some coff relaxing
   targets), and thus the original size needs to be kept to read the
   section multiple times. For output sections, rawsize holds the
   section size calculated on a previous linker relaxation pass. */
bfd_size_type rawsize;

/* The compressed size of the section in octets. */
bfd_size_type compressed_size;

/* If this section is going to be output, then this value is the
   offset in *bytes* into the output section of the first byte in the
   input section (byte ==> smallest addressable unit on the
   target). In most cases, if this was going to start at the
   100th octet (8-bit quantity) in the output section, this value
   would be 100. However, if the target byte size is 16 bits
   (bfd_octets_per_byte is "2"), this value would be 50. */
bfd_vma output_offset;

/* The output section through which to map on output. */
struct bfd_section *output_section;

/* If an input section, a pointer to a vector of relocation
   records for the data in this section. */
struct reloc_cache_entry *relocation;

/* If an output section, a pointer to a vector of pointers to
   relocation records for the data in this section. */
struct reloc_cache_entry **orelocation;
```



```
/* The number of relocation records in one of the above. */
unsigned reloc_count;

/* The alignment requirement of the section, as an exponent of 2 -
   e.g., 3 aligns to 2^3 (or 8). */
unsigned int alignment_power;

/* Information below is back end specific - and not always used
   or updated. */

/* File position of section data. */
file_ptr filepos;

/* File position of relocation info. */
file_ptr rel_filepos;

/* File position of line data. */
file_ptr line_filepos;

/* Pointer to data for applications. */
void *userdata;

/* If the SEC_IN_MEMORY flag is set, this points to the actual
   contents. */
bfd_byte *contents;

/* Attached line number information. */
alient *lineno;

/* Number of line number records. */
unsigned int lineno_count;

/* Entity size for merging purposes. */
unsigned int entsize;

/* Points to the kept section if this section is a link-once section,
   and is discarded. */
struct bfd_section *kept_section;

/* When a section is being output, this value changes as more
   linenumbers are written out. */
file_ptr moving_line_filepos;

/* What the section number is in the target world. */
int target_index;
```

```

void *used_by_bfd;

/* If this is a constructor section then here is a list of the
   relocations created to relocate items within it. */
struct relent_chain *constructor_chain;

/* The BFD which owns the section. */
bfd *owner;

/* A symbol which points at this section only. */
struct bfd_symbol *symbol;
struct bfd_symbol **symbol_ptr_ptr;

/* Early in the link process, map_head and map_tail are used to build
   a list of input sections attached to an output section. Later,
   output sections use these fields for a list of bfd_link_order
   structs. The linked_to_symbol_name field is for ELF assembler
   internal use. */
union {
  struct bfd_link_order *link_order;
  struct bfd_section *s;
  const char *linked_to_symbol_name;
} map_head, map_tail;

/* Points to the output section this section is already assigned to,
   if any. This is used when support for non-contiguous memory
   regions is enabled. */
struct bfd_section *already_assigned;

/* Explicitly specified section type, if non-zero. */
unsigned int type;
} asection;

```

2.5.5 Section prototypes

These are the functions exported by the section handling part of BFD.

2.5.5.1 bfd_section_list_clear

```
void bfd_section_list_clear (bfd *);
```

[Function]

Clears the section list, and also resets the section count and hash table entries.

2.5.5.2 bfd_get_section_by_name

```
asection *bfd_get_section_by_name (bfd *abfd, const char [Function]
    *name);
```

Return the most recently created section attached to *abfd* named *name*. Return NULL if no such section exists.

2.5.5.3 bfd_get_next_section_by_name

```
asection *bfd_get_next_section_by_name (bfd *ibfd, [Function]
    asection *sec);
```

Given *sec* is a section returned by `bfd_get_section_by_name`, return the next most recently created section attached to the same BFD with the same name, or if no such section exists in the same BFD and IBFD is non-NULL, the next section with the same name in any input BFD following IBFD. Return NULL on finding no section.

2.5.5.4 bfd_get_linker_section

```
asection *bfd_get_linker_section (bfd *abfd, const char [Function]
    *name);
```

Return the linker created section attached to *abfd* named *name*. Return NULL if no such section exists.

2.5.5.5 bfd_get_section_by_name_if

```
asection *bfd_get_section_by_name_if (bfd *abfd, const [Function]
    char *name, bool (*func) (bfd *abfd, asection *sect, void
    *obj), void *obj);
```

Call the provided function *func* for each section attached to the BFD *abfd* whose name matches *name*, passing *obj* as an argument. The function will be called as if by

```
func (abfd, the_section, obj);
```

It returns the first section for which *func* returns true, otherwise NULL.

2.5.5.6 bfd_get_unique_section_name

```
char *bfd_get_unique_section_name (bfd *abfd, const char [Function]
    *templat, int *count);
```

Invent a section name that is unique in *abfd* by tacking a dot and a digit suffix onto the original *templat*. If *count* is non-NULL, then it specifies the first number tried as a suffix to generate a unique name. The value pointed to by *count* will be incremented in this case.

2.5.5.7 bfd_make_section_old_way

```
asection *bfd_make_section_old_way (bfd *abfd, const char [Function]
    *name);
```

Create a new empty section called *name* and attach it to the end of the chain of sections for the BFD *abfd*. An attempt to create a section with a name which is already in use returns its pointer without changing the section chain.

It has the funny name since this is the way it used to be before it was rewritten....

Possible errors are:

- `bfd_error_invalid_operation` - If output has already started for this BFD.
- `bfd_error_no_memory` - If memory allocation fails.

2.5.5.8 `bfd_make_section_anyway_with_flags`

```
asection *bfd_make_section_anyway_with_flags (bfd *abfd,          [Function]
      const char *name, flagword flags);
```

Create a new empty section called *name* and attach it to the end of the chain of sections for *abfd*. Create a new section even if there is already a section with that name. Also set the attributes of the new section to the value *flags*.

Return NULL and set `bfd_error` on error; possible errors are:

- `bfd_error_invalid_operation` - If output has already started for *abfd*.
- `bfd_error_no_memory` - If memory allocation fails.

2.5.5.9 `bfd_make_section_anyway`

```
asection *bfd_make_section_anyway (bfd *abfd, const char        [Function]
      *name);
```

Create a new empty section called *name* and attach it to the end of the chain of sections for *abfd*. Create a new section even if there is already a section with that name.

Return NULL and set `bfd_error` on error; possible errors are:

- `bfd_error_invalid_operation` - If output has already started for *abfd*.
- `bfd_error_no_memory` - If memory allocation fails.

2.5.5.10 `bfd_make_section_with_flags`

```
asection *bfd_make_section_with_flags (bfd *, const char        [Function]
      *name, flagword flags);
```

Like `bfd_make_section_anyway`, but return NULL (without calling `bfd_set_error ()`) without changing the section chain if there is already a section named *name*. Also set the attributes of the new section to the value *flags*. If there is an error, return NULL and set `bfd_error`.

2.5.5.11 `bfd_make_section`

```
asection *bfd_make_section (bfd *, const char *name);          [Function]
```

Like `bfd_make_section_anyway`, but return NULL (without calling `bfd_set_error ()`) without changing the section chain if there is already a section named *name*. If there is an error, return NULL and set `bfd_error`.

2.5.5.12 `bfd_set_section_flags`

```
bool bfd_set_section_flags (asection *sec, flagword flags);    [Function]
```

Set the attributes of the section *sec* to the value *flags*. Return TRUE on success, FALSE on error. Possible error returns are:

- `bfd_error_invalid_operation` - The section cannot have one or more of the attributes requested. For example, a `.bss` section in `a.out` may not have the `SEC_HAS_CONTENTS` field set.

2.5.5.13 `bfd_rename_section`

```
void bfd_rename_section (asection *sec, const char          [Function]
                        *newname);
```

Rename section *sec* to *newname*.

2.5.5.14 `bfd_map_over_sections`

```
void bfd_map_over_sections (bfd *abfd, void (*func) (bfd      [Function]
                *abfd, asection *sect, void *obj), void *obj);
```

Call the provided function *func* for each section attached to the BFD *abfd*, passing *obj* as an argument. The function will be called as if by

```
func (abfd, the_section, obj);
```

This is the preferred method for iterating over sections; an alternative would be to use a loop:

```
asection *p;
for (p = abfd->sections; p != NULL; p = p->next)
    func (abfd, p, ...)
```

2.5.5.15 `bfd_sections_find_if`

```
asection *bfd_sections_find_if (bfd *abfd, bool              [Function]
                               (*operation) (bfd *abfd, asection *sect, void *obj), void
                               *obj);
```

Call the provided function *operation* for each section attached to the BFD *abfd*, passing *obj* as an argument. The function will be called as if by

```
operation (abfd, the_section, obj);
```

It returns the first section for which *operation* returns true.

2.5.5.16 `bfd_set_section_size`

```
bool bfd_set_section_size (asection *sec, bfd_size_type     [Function]
                          val);
```

Set *sec* to the size *val*. If the operation is ok, then `TRUE` is returned, else `FALSE`.

Possible error returns:

- `bfd_error_invalid_operation` - Writing has started to the BFD, so setting the size is invalid.

2.5.5.17 bfd_set_section_contents

```
bool bfd_set_section_contents (bfd *abfd, asection [Function]
                             *section, const void *data, file_ptr offset, bfd_size_type
                             count);
```

Sets the contents of the section *section* in BFD *abfd* to the data starting in memory at *location*. The data is written to the output section starting at offset *offset* for *count* octets.

Normally **TRUE** is returned, but **FALSE** is returned if there was an error. Possible error returns are:

- `bfd_error_no_contents` - The output section does not have the `SEC_HAS_CONTENTS` attribute, so nothing can be written to it.
- `bfd_error_bad_value` - The section is unable to contain all of the data.
- `bfd_error_invalid_operation` - The BFD is not writeable.
- and some more too.

This routine is front end to the back end function `_bfd_set_section_contents`.

2.5.5.18 bfd_get_section_contents

```
bool bfd_get_section_contents (bfd *abfd, asection [Function]
                              *section, void *location, file_ptr offset, bfd_size_type
                              count);
```

Read data from *section* in BFD *abfd* into memory starting at *location*. The data is read at an offset of *offset* from the start of the input section, and is read for *count* bytes.

If the contents of a constructor with the `SEC_CONSTRUCTOR` flag set are requested or if the section does not have the `SEC_HAS_CONTENTS` flag set, then the *location* is filled with zeroes. If no errors occur, **TRUE** is returned, else **FALSE**.

2.5.5.19 bfd_malloc_and_get_section

```
bool bfd_malloc_and_get_section (bfd *abfd, asection [Function]
                                 *section, bfd_byte **buf);
```

Read all data from *section* in BFD *abfd* into a buffer, **buf*, malloc'd by this function. Return **true** on success, **false** on failure in which case **buf* will be **NULL**.

2.5.5.20 bfd_copy_private_section_data

```
bool bfd_copy_private_section_data (bfd *ibfd, asection [Function]
                                    *isec, bfd *obfd, asection *osec);
```

Copy private section information from *isec* in the BFD *ibfd* to the section *osec* in the BFD *obfd*. Return **TRUE** on success, **FALSE** on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *osec*.

```
#define bfd_copy_private_section_data(ibfd, isection, obfd, osection) \
    BFD_SEND (obfd, _bfd_copy_private_section_data, \
              (ibfd, isection, obfd, osection))
```

2.5.5.21 `bfd_generic_is_group_section`

```
bool bfd_generic_is_group_section (bfd *, const asection      [Function]
                                   *sec);
```

Returns TRUE if *sec* is a member of a group.

2.5.5.22 `bfd_generic_group_name`

```
const char *bfd_generic_group_name (bfd *, const asection     [Function]
                                     *sec);
```

Returns group name if *sec* is a member of a group.

2.5.5.23 `bfd_generic_discard_group`

```
bool bfd_generic_discard_group (bfd *abfd, asection          [Function]
                                 *group);
```

Remove all members of *group* from the output.

2.5.5.24 `_bfd_section_size_insane`

```
bool _bfd_section_size_insane (bfd *abfd, asection *sec);    [Function]
```

Returns true if the given section has a size that indicates it cannot be read from file. Return false if the size is OK or* this function can't say one way or the other.

2.6 Symbols

BFD tries to maintain as much symbol information as it can when it moves information from file to file. BFD passes information to applications through the `asymbol` structure. When the application requests the symbol table, BFD reads the table in the native form and translates parts of it into the internal format. To maintain more than the information passed to applications, some targets keep some information “behind the scenes” in a structure only the particular back end knows about. For example, the `coff` back end keeps the original symbol table structure as well as the canonical structure when a BFD is read in. On output, the `coff` back end can reconstruct the output symbol table so that no information is lost, even information unique to `coff` which BFD doesn't know or understand. If a `coff` symbol table were read, but were written through an `a.out` back end, all the `coff` specific information would be lost. The symbol table of a BFD is not necessarily read in until a `canonicalize` request is made. Then the BFD back end fills in a table provided by the application with pointers to the canonical information. To output symbols, the application provides BFD with a table of pointers to pointers to `asymbols`. This allows applications like the linker to output a symbol as it was read, since the “behind the scenes” information will be still available.

2.6.1 Reading symbols

There are two stages to reading a symbol table from a BFD: allocating storage, and the actual reading process. This is an excerpt from an application which reads the symbol table:

```
long storage_needed;
asymbol **symbol_table;
```

```

long number_of_symbols;
long i;

storage_needed = bfd_get_symtab_upper_bound (abfd);

if (storage_needed < 0)
    FAIL

if (storage_needed == 0)
    return;

symbol_table = xmalloc (storage_needed);
...
number_of_symbols =
    bfd_canonicalize_symtab (abfd, symbol_table);

if (number_of_symbols < 0)
    FAIL

for (i = 0; i < number_of_symbols; i++)
    process_symbol (symbol_table[i]);

```

All storage for the symbols themselves is in an objalloc connected to the BFD; it is freed when the BFD is closed.

2.6.2 Writing symbols

Writing of a symbol table is automatic when a BFD open for writing is closed. The application attaches a vector of pointers to pointers to symbols to the BFD being written, and fills in the symbol count. The close and cleanup code reads through the table provided and performs all the necessary operations. The BFD output code must always be provided with an “owned” symbol: one which has come from another BFD, or one which has been created using `bfd_make_empty_symbol`. Here is an example showing the creation of a symbol table with only one element:

```

#include "sysdep.h"
#include "bfd.h"
int main (void)
{
    bfd *abfd;
    asymbol *ptrs[2];
    asymbol *new;

    abfd = bfd_openw ("foo","a.out-sunos-big");
    bfd_set_format (abfd, bfd_object);
    new = bfd_make_empty_symbol (abfd);
    new->name = "dummy_symbol";
    new->section = bfd_make_section_old_way (abfd, ".text");
    new->flags = BSF_GLOBAL;

```



```

    new->value = 0x12345;

    ptrs[0] = new;
    ptrs[1] = 0;

    bfd_set_symtab (abfd, ptrs, 1);
    bfd_close (abfd);
    return 0;
}

./makesym
nm foo
00012345 A dummy_symbol

```

Many formats cannot represent arbitrary symbol information; for instance, the `a.out` object format does not allow an arbitrary number of sections. A symbol pointing to a section which is not one of `.text`, `.data` or `.bss` cannot be described.

2.6.3 Mini Symbols

Mini symbols provide read-only access to the symbol table. They use less memory space, but require more time to access. They can be useful for tools like `nm` or `objdump`, which may have to handle symbol tables of extremely large executables.

The `bfd_read_minisymbols` function will read the symbols into memory in an internal form. It will return a `void *` pointer to a block of memory, a symbol count, and the size of each symbol. The pointer is allocated using `malloc`, and should be freed by the caller when it is no longer needed.

The function `bfd_minisymbol_to_symbol` will take a pointer to a minisymbol, and a pointer to a structure returned by `bfd_make_empty_symbol`, and return a `asymbol` structure. The return value may or may not be the same as the value from `bfd_make_empty_symbol` which was passed in.

2.6.4 typedef asymbol

An `asymbol` has the form:

```

typedef struct bfd_symbol
{
    /* A pointer to the BFD which owns the symbol. This information
       is necessary so that a back end can work out what additional
       information (invisible to the application writer) is carried
       with the symbol.

       This field is almost redundant, since you can use section->owner
       instead, except that some symbols point to the global sections
       bfd_{abs,com,und}_section. This could be fixed by making
       these globals be per-bfd (or per-target-flavor). FIXME. */
    struct bfd *the_bfd; /* Use bfd_asymbol_bfd(sym) to access this field. */

    /* The text of the symbol. The name is left alone, and not copied; the

```

```
    application may not alter it.  */
    const char *name;

    /* The value of the symbol.  This really should be a union of a
       numeric value with a pointer, since some flags indicate that
       a pointer to another symbol is stored here.  */
    symvalue value;

    /* Attributes of a symbol.  */
#define BSF_NO_FLAGS          0

    /* The symbol has local scope; static in C. The value
       is the offset into the section of the data.  */
#define BSF_LOCAL            (1 << 0)

    /* The symbol has global scope; initialized data in C. The
       value is the offset into the section of the data.  */
#define BSF_GLOBAL          (1 << 1)

    /* The symbol has global scope and is exported. The value is
       the offset into the section of the data.  */
#define BSF_EXPORT          BSF_GLOBAL /* No real difference.  */

    /* A normal C symbol would be one of:
       BSF_LOCAL, BSF_UNDEFINED or BSF_GLOBAL.  */

    /* The symbol is a debugging record. The value has an arbitrary
       meaning, unless BSF_DEBUGGING_RELOC is also set.  */
#define BSF_DEBUGGING      (1 << 2)

    /* The symbol denotes a function entry point.  Used in ELF,
       perhaps others someday.  */
#define BSF_FUNCTION       (1 << 3)

    /* Used by the linker.  */
#define BSF_KEEP           (1 << 5)

    /* An ELF common symbol.  */
#define BSF_ELF_COMMON     (1 << 6)

    /* A weak global symbol, overridable without warnings by
       a regular global symbol of the same name.  */
#define BSF_WEAK           (1 << 7)

    /* This symbol was created to point to a section, e.g. ELF's
       STT_SECTION symbols.  */
#define BSF_SECTION_SYM    (1 << 8)
```

```
/* The symbol used to be a common symbol, but now it is
   allocated. */
#define BSF_OLD_COMMON          (1 << 9)

/* In some files the type of a symbol sometimes alters its
   location in an output file - ie in coff a ISFCN symbol
   which is also C_EXT symbol appears where it was
   declared and not at the end of a section. This bit is set
   by the target BFD part to convey this information. */
#define BSF_NOT_AT_END         (1 << 10)

/* Signal that the symbol is the label of constructor section. */
#define BSF_CONSTRUCTOR        (1 << 11)

/* Signal that the symbol is a warning symbol. The name is a
   warning. The name of the next symbol is the one to warn about;
   if a reference is made to a symbol with the same name as the next
   symbol, a warning is issued by the linker. */
#define BSF_WARNING            (1 << 12)

/* Signal that the symbol is indirect. This symbol is an indirect
   pointer to the symbol with the same name as the next symbol. */
#define BSF_INDIRECT           (1 << 13)

/* BSF_FILE marks symbols that contain a file name. This is used
   for ELF STT_FILE symbols. */
#define BSF_FILE                (1 << 14)

/* Symbol is from dynamic linking information. */
#define BSF_DYNAMIC             (1 << 15)

/* The symbol denotes a data object. Used in ELF, and perhaps
   others someday. */
#define BSF_OBJECT              (1 << 16)

/* This symbol is a debugging symbol. The value is the offset
   into the section of the data. BSF_DEBUGGING should be set
   as well. */
#define BSF_DEBUGGING_RELOC     (1 << 17)

/* This symbol is thread local. Used in ELF. */
#define BSF_THREAD_LOCAL        (1 << 18)

/* This symbol represents a complex relocation expression,
   with the expression tree serialized in the symbol name. */
#define BSF_RELC                (1 << 19)
```

```

    /* This symbol represents a signed complex relocation expression,
       with the expression tree serialized in the symbol name. */
#define BSF_SRELC                (1 << 20)

    /* This symbol was created by bfd_get_synthetic_symtab. */
#define BSF_SYNTHETIC            (1 << 21)

    /* This symbol is an indirect code object. Unrelated to BSF_INDIRECT.
       The dynamic linker will compute the value of this symbol by
       calling the function that it points to. BSF_FUNCTION must
       also be also set. */
#define BSF_GNU_INDIRECT_FUNCTION (1 << 22)
    /* This symbol is a globally unique data object. The dynamic linker
       will make sure that in the entire process there is just one symbol
       with this name and type in use. BSF_OBJECT must also be set. */
#define BSF_GNU_UNIQUE            (1 << 23)

    /* This section symbol should be included in the symbol table. */
#define BSF_SECTION_SYM_USED      (1 << 24)

    flagword flags;

    /* A pointer to the section to which this symbol is
       relative. This will always be non NULL, there are special
       sections for undefined and absolute symbols. */
    struct bfd_section *section;

    /* Back end special data. */
    union
    {
        void *p;
        bfd_vma i;
    }
    udata;
}
asymbol;

```

2.6.5 Symbol handling functions

2.6.5.1 `bfd_get_symtab_upper_bound`

Return the number of bytes required to store a vector of pointers to `asymbols` for all the symbols in the BFD `abfd`, including a terminal NULL pointer. If there are no symbols in the BFD, then return 0. If an error occurs, return -1.

```
#define bfd_get_symtab_upper_bound(abfd) \
```

```
BFD_SEND (abfd, _bfd_get_syntab_upper_bound, (abfd))
```

2.6.5.2 bfd_is_local_label

```
bool bfd_is_local_label (bfd *abfd, asymbol *sym);           [Function]
    Return TRUE if the given symbol sym in the BFD abfd is a compiler generated local
    label, else return FALSE.
```

2.6.5.3 bfd_is_local_label_name

```
bool bfd_is_local_label_name (bfd *abfd, const char *name); [Function]
    Return TRUE if a symbol with the name name in the BFD abfd is a compiler generated
    local label, else return FALSE. This just checks whether the name has the form of a
    local label.
```

```
#define bfd_is_local_label_name(abfd, name) \
    BFD_SEND (abfd, _bfd_is_local_label_name, (abfd, name))
```

2.6.5.4 bfd_is_target_special_symbol

```
bool bfd_is_target_special_symbol (bfd *abfd, asymbol      [Function]
    *sym);
    Return TRUE iff a symbol sym in the BFD abfd is something special to the particular
    target represented by the BFD. Such symbols should normally not be mentioned to
    the user.
```

```
#define bfd_is_target_special_symbol(abfd, sym) \
    BFD_SEND (abfd, _bfd_is_target_special_symbol, (abfd, sym))
```

2.6.5.5 bfd_canonicalize_syntab

Read the symbols from the BFD *abfd*, and fills in the vector *location* with pointers to the symbols and a trailing NULL. Return the actual number of symbol pointers, not including the NULL.

```
#define bfd_canonicalize_syntab(abfd, location) \
    BFD_SEND (abfd, _bfd_canonicalize_syntab, (abfd, location))
```

2.6.5.6 bfd_set_syntab

```
bool bfd_set_syntab (bfd *abfd, asymbol **location,        [Function]
    unsigned int count);
    Arrange that when the output BFD abfd is closed, the table location of count pointers
    to symbols will be written.
```

2.6.5.7 bfd_print_symbol_vandf

```
void bfd_print_symbol_vandf (bfd *abfd, void *file,          [Function]
                             asymbol *symbol);
```

Print the value and flags of the *symbol* supplied to the stream *file*.

2.6.5.8 bfd_make_empty_symbol

Create a new `asymbol` structure for the BFD *abfd* and return a pointer to it.

This routine is necessary because each back end has private information surrounding the `asymbol`. Building your own `asymbol` and pointing to it will not create the private information, and will cause problems later on.

```
#define bfd_make_empty_symbol(abfd) \
    BFD_SEND (abfd, _bfd_make_empty_symbol, (abfd))
```

2.6.5.9 _bfd_generic_make_empty_symbol

```
asymbol *_bfd_generic_make_empty_symbol (bfd *);          [Function]
```

Create a new `asymbol` structure for the BFD *abfd* and return a pointer to it. Used by core file routines, binary back-end and anywhere else where no private info is needed.

2.6.5.10 bfd_make_debug_symbol

Create a new `asymbol` structure for the BFD *abfd*, to be used as a debugging symbol.

```
#define bfd_make_debug_symbol(abfd) \
    BFD_SEND (abfd, _bfd_make_debug_symbol, (abfd))
```

2.6.5.11 bfd_decode_symclass

```
int bfd_decode_symclass (asymbol *symbol);                [Function]
```

Return a character corresponding to the symbol class of *symbol*, or '?' for an unknown class.

2.6.5.12 bfd_is_undefined_symclass

```
bool bfd_is_undefined_symclass (int symclass);            [Function]
```

Returns non-zero if the class symbol returned by `bfd_decode_symclass` represents an undefined symbol. Returns zero otherwise.

2.6.5.13 bfd_symbol_info

```
void bfd_symbol_info (asymbol *symbol, symbol_info *ret); [Function]
```

Fill in the basic info about symbol that nm needs. Additional info may be added by the back-ends after calling this function.

2.6.5.14 bfd_copy_private_symbol_data

```
bool bfd_copy_private_symbol_data (bfd *ibfd, asymbol [Function]
    *isym, bfd *obfd, asymbol *osym);
```

Copy private symbol information from *isym* in the BFD *ibfd* to the symbol *osym* in the BFD *obfd*. Return TRUE on success, FALSE on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *osym*.

```
#define bfd_copy_private_symbol_data(ibfd, isymbol, obfd, osymbol) \
    BFD_SEND (obfd, _bfd_copy_private_symbol_data, \
              (ibfd, isymbol, obfd, osymbol))
```

2.7 Archives

An archive (or library) is just another BFD. It has a symbol table, although there's not much a user program will do with it.

The big difference between an archive BFD and an ordinary BFD is that the archive doesn't have sections. Instead it has a chain of BFDs that are considered its contents. These BFDs can be manipulated like any other. The BFDs contained in an archive opened for reading will all be opened for reading. You may put either input or output BFDs into an archive opened for output; they will be handled correctly when the archive is closed.

Use `bfd_openr_next_archived_file` to step through the contents of an archive opened for input. You don't have to read the entire archive if you don't want to! Read it until you find what you want.

A BFD returned by `bfd_openr_next_archived_file` can be closed manually with `bfd_close`. If you do not close it, then a second iteration through the members of an archive may return the same BFD. If you close the archive BFD, then all the member BFDs will automatically be closed as well.

Archive contents of output BFDs are chained through the `archive_next` pointer in a BFD. The first one is findable through the `archive_head` slot of the archive. Set it with `bfd_set_archive_head` (q.v.). A given BFD may be in only one open output archive at a time.

As expected, the BFD archive code is more general than the archive code of any given environment. BFD archives may contain files of different formats (e.g., a.out and coff) and even different architectures. You may even place archives recursively into archives!

This can cause unexpected confusion, since some archive formats are more expressive than others. For instance, Intel COFF archives can preserve long filenames; SunOS a.out archives cannot. If you move a file from the first to the second format and back again, the filename may be truncated. Likewise, different a.out environments have different conventions as to how they truncate filenames, whether they preserve directory names in filenames, etc. When interoperating with native tools, be sure your files are homogeneous.

Beware: most of these formats do not react well to the presence of spaces in filenames. We do the best we can, but can't always handle this case due to restrictions in the format of archives. Many Unix utilities are braindead in regards to spaces and such in filenames anyway, so this shouldn't be much of a restriction.

Archives are supported in BFD in `archive.c`.

2.7.1 Archive functions

2.7.1.1 `bfd_get_next_mapent`

```
symindex bfd_get_next_mapent (bfd *abfd, symindex          [Function]
                             previous, carsym **sym);
```

Step through archive *abfd*'s symbol table (if it has one). Successively update *sym* with the next symbol's information, returning that symbol's (internal) index into the symbol table.

Supply `BFD_NO_MORE_SYMBOLS` as the *previous* entry to get the first one; returns `BFD_NO_MORE_SYMBOLS` when you've already got the last one.

A *carsym* is a canonical archive symbol. The only user-visible element is its name, a null-terminated string.

2.7.1.2 `bfd_set_archive_head`

```
bool bfd_set_archive_head (bfd *output, bfd *new_head);      [Function]
Set the head of the chain of BFDs contained in the archive output to new_head.
```

2.7.1.3 `bfd_openr_next_archived_file`

```
bfd *bfd_openr_next_archived_file (bfd *archive, bfd        [Function]
                                   *previous);
```

Provided a BFD, *archive*, containing an archive and NULL, open an input BFD on the first contained element and returns that. Subsequent calls should pass the archive and the previous return value to return a created BFD to the next contained element. NULL is returned when there are no more. Note - if you want to process the bfd returned by this call be sure to call `bfd_check_format()` on it first.

2.8 File formats

A format is a BFD concept of high level file contents type. The formats supported by BFD are:

- `bfd_object`

The BFD may contain data, symbols, relocations and debug info.

- `bfd_archive`

The BFD contains other BFDs and an optional index.

- `bfd_core`

The BFD contains the result of an executable core dump.

2.8.1 File format functions

2.8.1.1 bfd_check_format

`bool bfd_check_format (bfd *abfd, bfd_format format);` [Function]

Verify if the file attached to the BFD *abfd* is compatible with the format *format* (i.e., one of `bfd_object`, `bfd_archive` or `bfd_core`).

If the BFD has been set to a specific target before the call, only the named target and format combination is checked. If the target has not been set, or has been set to `default`, then all the known target backends is interrogated to determine a match. If the default target matches, it is used. If not, exactly one target must recognize the file, or an error results.

The function returns `TRUE` on success, otherwise `FALSE` with one of the following error codes:

- `bfd_error_invalid_operation` - if *format* is not one of `bfd_object`, `bfd_archive` or `bfd_core`.
- `bfd_error_system_call` - if an error occurred during a read - even some file mismatches can cause `bfd_error_system_calls`.
- `file_not_recognised` - none of the backends recognised the file format.
- `bfd_error_file_ambiguously_recognized` - more than one backend recognised the file format.

2.8.1.2 bfd_check_format_matches

`bool bfd_check_format_matches (bfd *abfd, bfd_format format, char ***matching);` [Function]

Like `bfd_check_format`, except when it returns `FALSE` with `bfd_errno` set to `bfd_error_file_ambiguously_recognized`. In that case, if *matching* is not `NULL`, it will be filled in with a `NULL`-terminated list of the names of the formats that matched, allocated with `malloc`. Then the user may choose a format and try again.

When done with the list that *matching* points to, the caller should free it.

2.8.1.3 bfd_set_format

`bool bfd_set_format (bfd *abfd, bfd_format format);` [Function]

This function sets the file format of the BFD *abfd* to the format *format*. If the target set in the BFD does not support the format requested, the format is invalid, or the BFD is not open for writing, then an error occurs.

2.8.1.4 bfd_format_string

`const char *bfd_format_string (bfd_format format);` [Function]

Return a pointer to a const string `invalid`, `object`, `archive`, `core`, or `unknown`, depending upon the value of *format*.

2.9 Relocations

BFD maintains relocations in much the same way it maintains symbols: they are left alone until required, then read in en-masse and translated into an internal form. A common routine `bfd_perform_relocation` acts upon the canonical form to do the fixup.

Relocations are maintained on a per section basis, while symbols are maintained on a per BFD basis.

All that a back end has to do to fit the BFD interface is to create a `struct reloc_cache_entry` for each relocation in a particular section, and fill in the right bits of the structures.

2.9.1 typedef arelent

This is the structure of a relocation entry:

```
struct reloc_cache_entry
{
    /* A pointer into the canonical table of pointers.  */
    struct bfd_symbol **sym_ptr_ptr;

    /* offset in section.  */
    bfd_size_type address;

    /* addend for relocation value.  */
    bfd_vma addend;

    /* Pointer to how to perform the required relocation.  */
    reloc_howto_type *howto;
};
```

Here is a description of each of the fields within an `arelent`:

- `sym_ptr_ptr`

The symbol table pointer points to a pointer to the symbol associated with the relocation request. It is the pointer into the table returned by the back end's `canonicalize_syntab` action. See Section 2.6 [Symbols], page 38. The symbol is referenced through a pointer to a pointer so that tools like the linker can fix up all the symbols of the same name by modifying only one pointer. The relocation routine looks in the symbol and uses the base of the section the symbol is attached to and the value of the symbol as the initial relocation offset. If the symbol pointer is zero, then the section provided is looked up.

- `address`

The `address` field gives the offset in bytes from the base of the section data which owns the relocation record to the first byte of relocatable information. The actual data relocated will be relative to this point; for example, a relocation type which modifies the bottom two bytes of a four byte word would not touch the first byte pointed to in a big endian world.

- `addend`

The `addend` is a value provided by the back end to be added (!) to the relocation offset. Its interpretation is dependent upon the `howto`. For example, on the 68k the code:

```
char foo[];
main()
{
    return foo[0x12345678];
}
```

```
    }
```

Could be compiled into:

```
    linkw fp,#-4
    moveb @#12345678,d0
    extbl d0
    unlk fp
    rts
```

This could create a reloc pointing to `foo`, but leave the offset in the data, something like:

```
RELOCATION RECORDS FOR [.text]:
offset  type      value
00000006 32          _foo

00000000 4e56 fffc          ; linkw fp,#-4
00000004 1039 1234 5678      ; moveb @#12345678,d0
0000000a 49c0          ; extbl d0
0000000c 4e5e          ; unlk fp
0000000e 4e75          ; rts
```

Using `coff` and an 88k, some instructions don't have enough space in them to represent the full address range, and pointers have to be loaded in two parts. So you'd get something like:

```
    or.u   r13,r0,hi16(_foo+0x12345678)
    ld.b   r2,r13,lo16(_foo+0x12345678)
    jmp    r1
```

This should create two relocs, both pointing to `_foo`, and with `0x12340000` in their addend field. The data would consist of:

```
RELOCATION RECORDS FOR [.text]:
offset  type      value
00000002 HVRT16    _foo+0x12340000
00000006 LVRT16    _foo+0x12340000

00000000 5da05678          ; or.u r13,r0,0x5678
00000004 1c4d5678          ; ld.b r2,r13,0x5678
00000008 f400c001          ; jmp r1
```

The relocation routine digs out the value from the data, adds it to the addend to get the original offset, and then adds the value of `_foo`. Note that all 32 bits have to be kept around somewhere, to cope with carry from bit 15 to bit 16.

One further example is the `sparc` and the `a.out` format. The `sparc` has a similar problem to the 88k, in that some instructions don't have room for an entire offset, but on the `sparc` the parts are created in odd sized lumps. The designers of the `a.out` format chose to not use the data within the section for storing part of the offset; all the offset is kept within the reloc. Anything in the data should be ignored.

```
    save %sp,-112,%sp
    sethi %hi(_foo+0x12345678),%g2
    ldsb [%g2+%lo(_foo+0x12345678)],%i0
    ret
```

```
        restore
```

Both relocs contain a pointer to `foo`, and the offsets contain junk.

```
RELOCATION RECORDS FOR [.text]:
offset  type      value
00000004 HI22      _foo+0x12345678
00000008 L010      _foo+0x12345678

00000000 9de3bf90      ; save %sp,-112,%sp
00000004 05000000      ; sethi %hi(_foo+0),%g2
00000008 f048a000      ; ldsb [%g2+%lo(_foo+0)],%i0
0000000c 81c7e008      ; ret
00000010 81e80000      ; restore
```

- `howto`

The `howto` field can be imagined as a relocation instruction. It is a pointer to a structure which contains information on what to do with all of the other information in the reloc record and data section. A back end would normally have a relocation instruction set and turn relocations into pointers to the correct structure on input - but it would be possible to create each `howto` field on demand.

2.9.1.1 `enum complain_overflow`

Indicates what sort of overflow checking should be done when performing a relocation.

```
enum complain_overflow
{
    /* Do not complain on overflow.  */
    complain_overflow_dont,

    /* Complain if the value overflows when considered as a signed
       number one bit larger than the field.  ie. A bitfield of N bits
       is allowed to represent -2**n to 2**n-1.  */
    complain_overflow_bitfield,

    /* Complain if the value overflows when considered as a signed
       number.  */
    complain_overflow_signed,

    /* Complain if the value overflows when considered as an
       unsigned number.  */
    complain_overflow_unsigned
};
```

2.9.1.2 `reloc_howto_type`

The `reloc_howto_type` is a structure which contains all the information that libbfd needs to know to tie up a back end's data.

```
struct reloc_howto_struct
```

```
{
/* The type field has mainly a documentary use - the back end can
   do what it wants with it, though normally the back end's idea of
   an external reloc number is stored in this field. */
unsigned int type;

/* The size of the item to be relocated in bytes. */
unsigned int size:4;

/* The number of bits in the field to be relocated. This is used
   when doing overflow checking. */
unsigned int bitsize:7;

/* The value the final relocation is shifted right by. This drops
   unwanted data from the relocation. */
unsigned int rightshift:6;

/* The bit position of the reloc value in the destination.
   The relocated value is left shifted by this amount. */
unsigned int bitpos:6;

/* What type of overflow error should be checked for when
   relocating. */
ENUM_BITFIELD (complain_overflow) complain_on_overflow:2;

/* The relocation value should be negated before applying. */
unsigned int negate:1;

/* The relocation is relative to the item being relocated. */
unsigned int pc_relative:1;

/* Some formats record a relocation addend in the section contents
   rather than with the relocation. For ELF formats this is the
   distinction between USE_REL and USE_RELA (though the code checks
   for USE_REL == 1/0). The value of this field is TRUE if the
   addend is recorded with the section contents; when performing a
   partial link (ld -r) the section contents (the data) will be
   modified. The value of this field is FALSE if addends are
   recorded with the relocation (in arelent.addend); when performing
   a partial link the relocation will be modified.
   All relocations for all ELF USE_RELA targets should set this field
   to FALSE (values of TRUE should be looked on with suspicion).
   However, the converse is not true: not all relocations of all ELF
   USE_REL targets set this field to TRUE. Why this is so is peculiar
   to each particular target. For relocs that aren't used in partial
   links (e.g. GOT stuff) it doesn't matter what this is set to. */
unsigned int partial_inplace:1;
```

```

/* When some formats create PC relative instructions, they leave
   the value of the pc of the place being relocated in the offset
   slot of the instruction, so that a PC relative relocation can
   be made just by adding in an ordinary offset (e.g., sun3 a.out).
   Some formats leave the displacement part of an instruction
   empty (e.g., ELF); this flag signals the fact. */
unsigned int pcrel_offset:1;

/* Whether bfd_install_relocation should just install the addend,
   or should follow the practice of some older object formats and
   install a value including the symbol. */
unsigned int install_addend:1;

/* src_mask selects the part of the instruction (or data) to be used
   in the relocation sum.  If the target relocations don't have an
   addend in the reloc, eg. ELF USE_REL, src_mask will normally equal
   dst_mask to extract the addend from the section contents.  If
   relocations do have an addend in the reloc, eg. ELF USE_RELA, this
   field should normally be zero.  Non-zero values for ELF USE_RELA
   targets should be viewed with suspicion as normally the value in
   the dst_mask part of the section contents should be ignored. */
bfd_vma src_mask;

/* dst_mask selects which parts of the instruction (or data) are
   replaced with a relocated value. */
bfd_vma dst_mask;

/* If this field is non null, then the supplied function is
   called rather than the normal function.  This allows really
   strange relocation methods to be accommodated. */
bfd_reloc_status_type (*special_function)
  (bfd *, arelent *, struct bfd_symbol *, void *, asection *,
   bfd *, char **);

/* The textual name of the relocation type. */
const char *name;
};

```

2.9.1.3 The HOWTO Macro

The HOWTO macro fills in a `reloc.howto_type` (a typedef for `const struct reloc.howto_struct`).

```

#define HOWTO_INSTALL_ADDEND 0
#define HOWTO_RSIZE(sz) ((sz) < 0 ? -(sz) : (sz))
#define HOWTO(type, right, size, bits, pcrel, left, ovf, func, name, \

```

```

        inplace, src_mask, dst_mask, pcrel_off)           \
{ (unsigned) type, HOWTO_RSIZE (size), bits, right, left, ovf, \
  size < 0, pcrel, inplace, pcrel_off, HOWTO_INSTALL_ADDEND, \
  src_mask, dst_mask, func, name }

```

This is used to fill in an empty howto entry in an array.

```

#define EMPTY_HOWTO(C) \
    HOWTO ((C), 0, 1, 0, false, 0, complain_overflow_dont, NULL, \
          NULL, false, 0, 0, false)

static inline unsigned int
bfd_get_reloc_size (reloc_howto_type *howto)
{
    return howto->size;
}

```

2.9.1.4 arelent_chain

How relocs are tied together in an asection:

```

typedef struct relent_chain
{
    arelent relent;
    struct relent_chain *next;
}
arelent_chain;

```

2.9.1.5 bfd_check_overflow

```

bfd_reloc_status_type bfd_check_overflow (enum [Function]
    complain_overflow how, unsigned int bitsize, unsigned int
    rightshift, unsigned int addrsize, bfd_vma relocation);

```

Perform overflow checking on *relocation* which has *bitsize* significant bits and will be shifted right by *rightshift* bits, on a machine with addresses containing *addrsize* significant bits. The result is either of `bfd_reloc_ok` or `bfd_reloc_overflow`.

2.9.1.6 bfd_reloc_offset_in_range

```

bool bfd_reloc_offset_in_range (reloc_howto_type *howto, [Function]
    bfd *abfd, asection *section, bfd_size_type offset);

```

Returns TRUE if the reloc described by *HOWTO* can be applied at *OFFSET* octets in *SECTION*.

2.9.1.7 bfd_perform_relocation

```
bfd_reloc_status_type bfd_perform_relocation (bfd *abfd,      [Function]
      arelent *reloc_entry, void *data, asection *input_section,
      bfd *output_bfd, char **error_message);
```

If *output_bfd* is supplied to this function, the generated image will be relocatable; the relocations are copied to the output file after they have been changed to reflect the new state of the world. There are two ways of reflecting the results of partial linkage in an output file: by modifying the output data in place, and by modifying the relocation record. Some native formats (e.g., basic a.out and basic coff) have no way of specifying an addend in the relocation type, so the addend has to go in the output data. This is no big deal since in these formats the output data slot will always be big enough for the addend. Complex reloc types with addends were invented to solve just this problem. The *error_message* argument is set to an error message if this return `bfd_reloc_dangerous`.

2.9.1.8 bfd_install_relocation

```
bfd_reloc_status_type bfd_install_relocation (bfd *abfd,      [Function]
      arelent *reloc_entry, void *data, bfd_vma data_start,
      asection *input_section, char **error_message);
```

This looks remarkably like `bfd_perform_relocation`, except it does not expect that the section contents have been filled in. I.e., it's suitable for use when creating, rather than applying a relocation.

For now, this function should be considered reserved for the assembler.

2.9.2 The howto manager

When an application wants to create a relocation, but doesn't know what the target machine might call it, it can find out by using this bit of code.

2.9.2.1 bfd_reloc_code_real_type

The insides of a reloc code. The idea is that, eventually, there will be one enumerator for every type of relocation we ever do. Pass one of these values to `bfd_reloc_type_lookup`, and it'll return a howto pointer.

This does mean that the application must determine the correct enumerator value; you can't get a howto pointer from a random set of attributes.

Here are the possible values for `enum bfd_reloc_code_real`:

```
BFD_RELOC_64
BFD_RELOC_32
BFD_RELOC_26
BFD_RELOC_24
BFD_RELOC_16
BFD_RELOC_14
BFD_RELOC_8
```

Basic absolute relocations of N bits.

BFD_RELOC_64_PCREL
BFD_RELOC_32_PCREL
BFD_RELOC_24_PCREL
BFD_RELOC_16_PCREL
BFD_RELOC_12_PCREL
BFD_RELOC_8_PCREL

PC-relative relocations. Sometimes these are relative to the address of the relocation itself; sometimes they are relative to the start of the section containing the relocation. It depends on the specific target.

BFD_RELOC_32_SECREL
BFD_RELOC_16_SECIDX

Section relative relocations. Some targets need this for DWARF2.

BFD_RELOC_32_GOT_PCREL
BFD_RELOC_16_GOT_PCREL
BFD_RELOC_8_GOT_PCREL
BFD_RELOC_32_GOTOFF
BFD_RELOC_16_GOTOFF
BFD_RELOC_LO16_GOTOFF
BFD_RELOC_HI16_GOTOFF
BFD_RELOC_HI16_S_GOTOFF
BFD_RELOC_8_GOTOFF
BFD_RELOC_64_PLT_PCREL
BFD_RELOC_32_PLT_PCREL
BFD_RELOC_24_PLT_PCREL
BFD_RELOC_16_PLT_PCREL
BFD_RELOC_8_PLT_PCREL
BFD_RELOC_64_PLTOFF
BFD_RELOC_32_PLTOFF
BFD_RELOC_16_PLTOFF
BFD_RELOC_LO16_PLTOFF
BFD_RELOC_HI16_PLTOFF
BFD_RELOC_HI16_S_PLTOFF
BFD_RELOC_8_PLTOFF

For ELF.

BFD_RELOC_SIZE32
BFD_RELOC_SIZE64

Size relocations.

BFD_RELOC_68K_GLOB_DAT
BFD_RELOC_68K_JMP_SLOT
BFD_RELOC_68K_RELATIVE
BFD_RELOC_68K_TLS_GD32
BFD_RELOC_68K_TLS_GD16
BFD_RELOC_68K_TLS_GD8
BFD_RELOC_68K_TLS_LDM32

BFD_RELOC_68K_TLS_LDM16
 BFD_RELOC_68K_TLS_LDM8
 BFD_RELOC_68K_TLS_LD032
 BFD_RELOC_68K_TLS_LD016
 BFD_RELOC_68K_TLS_LD08
 BFD_RELOC_68K_TLS_IE32
 BFD_RELOC_68K_TLS_IE16
 BFD_RELOC_68K_TLS_IE8
 BFD_RELOC_68K_TLS_LE32
 BFD_RELOC_68K_TLS_LE16
 BFD_RELOC_68K_TLS_LE8

Relocations used by 68K ELF.

BFD_RELOC_32_BASEREL
 BFD_RELOC_16_BASEREL
 BFD_RELOC_LO16_BASEREL
 BFD_RELOC_HI16_BASEREL
 BFD_RELOC_HI16_S_BASEREL
 BFD_RELOC_8_BASEREL
 BFD_RELOC_RVA

Linkage-table relative.

BFD_RELOC_8_FFnn

Absolute 8-bit relocation, but used to form an address like 0xFFnn.

BFD_RELOC_32_PCREL_S2
 BFD_RELOC_16_PCREL_S2
 BFD_RELOC_23_PCREL_S2

These PC-relative relocations are stored as word displacements – i.e., byte displacements shifted right two bits. The 30-bit word displacement (`<<32_PCREL_S2>>` – 32 bits, shifted 2) is used on the SPARC. (SPARC tools generally refer to this as `<<WDISP30>>`.) The signed 16-bit displacement is used on the MIPS, and the 23-bit displacement is used on the Alpha.

BFD_RELOC_HI22
 BFD_RELOC_LO10

High 22 bits and low 10 bits of 32-bit value, placed into lower bits of the target word. These are used on the SPARC.

BFD_RELOC_GPREL16
 BFD_RELOC_GPREL32

For systems that allocate a Global Pointer register, these are displacements off that register. These relocation types are handled specially, because the value the register will have is decided relatively late.

BFD_RELOC_NONE
 BFD_RELOC_SPARC_WDISP22
 BFD_RELOC_SPARC22
 BFD_RELOC_SPARC13

```

BFD_RELOC_SPARC_GOT10
BFD_RELOC_SPARC_GOT13
BFD_RELOC_SPARC_GOT22
BFD_RELOC_SPARC_PC10
BFD_RELOC_SPARC_PC22
BFD_RELOC_SPARC_WPLT30
BFD_RELOC_SPARC_COPY
BFD_RELOC_SPARC_GLOB_DAT
BFD_RELOC_SPARC_JMP_SLOT
BFD_RELOC_SPARC_RELATIVE
BFD_RELOC_SPARC_UA16
BFD_RELOC_SPARC_UA32
BFD_RELOC_SPARC_UA64
BFD_RELOC_SPARC_GOTDATA_HIX22
BFD_RELOC_SPARC_GOTDATA_LOX10
BFD_RELOC_SPARC_GOTDATA_OP_HIX22
BFD_RELOC_SPARC_GOTDATA_OP_LOX10
BFD_RELOC_SPARC_GOTDATA_OP
BFD_RELOC_SPARC_JMP_IREL
BFD_RELOC_SPARC_IRELATIVE

```

SPARC ELF relocations. There is probably some overlap with other relocation types already defined.

```

BFD_RELOC_SPARC_BASE13
BFD_RELOC_SPARC_BASE22

```

I think these are specific to SPARC a.out (e.g., Sun 4).

```

BFD_RELOC_SPARC_64
BFD_RELOC_SPARC_10
BFD_RELOC_SPARC_11
BFD_RELOC_SPARC_OL010
BFD_RELOC_SPARC_HH22
BFD_RELOC_SPARC_HM10
BFD_RELOC_SPARC_LM22
BFD_RELOC_SPARC_PC_HH22
BFD_RELOC_SPARC_PC_HM10
BFD_RELOC_SPARC_PC_LM22
BFD_RELOC_SPARC_WDISP16
BFD_RELOC_SPARC_WDISP19
BFD_RELOC_SPARC_7
BFD_RELOC_SPARC_6
BFD_RELOC_SPARC_5
BFD_RELOC_SPARC_DISP64
BFD_RELOC_SPARC_PLT32
BFD_RELOC_SPARC_PLT64
BFD_RELOC_SPARC_HIX22
BFD_RELOC_SPARC_LOX10
BFD_RELOC_SPARC_H44

```

BFD_RELOC_SPARC_M44
BFD_RELOC_SPARC_L44
BFD_RELOC_SPARC_REGISTER
BFD_RELOC_SPARC_H34
BFD_RELOC_SPARC_SIZE32
BFD_RELOC_SPARC_SIZE64
BFD_RELOC_SPARC_WDISP10
 SPARC64 relocations

BFD_RELOC_SPARC_REV32
 SPARC little endian relocation

BFD_RELOC_SPARC_TLS_GD_HI22
BFD_RELOC_SPARC_TLS_GD_LO10
BFD_RELOC_SPARC_TLS_GD_ADD
BFD_RELOC_SPARC_TLS_GD_CALL
BFD_RELOC_SPARC_TLS_LDM_HI22
BFD_RELOC_SPARC_TLS_LDM_LO10
BFD_RELOC_SPARC_TLS_LDM_ADD
BFD_RELOC_SPARC_TLS_LDM_CALL
BFD_RELOC_SPARC_TLS_LDO_HIX22
BFD_RELOC_SPARC_TLS_LDO_LOX10
BFD_RELOC_SPARC_TLS_LDO_ADD
BFD_RELOC_SPARC_TLS_IE_HI22
BFD_RELOC_SPARC_TLS_IE_LO10
BFD_RELOC_SPARC_TLS_IE_LD
BFD_RELOC_SPARC_TLS_IE_LDX
BFD_RELOC_SPARC_TLS_IE_ADD
BFD_RELOC_SPARC_TLS_LE_HIX22
BFD_RELOC_SPARC_TLS_LE_LOX10
BFD_RELOC_SPARC_TLS_DTPMOD32
BFD_RELOC_SPARC_TLS_DTPMOD64
BFD_RELOC_SPARC_TLS_DTPOFF32
BFD_RELOC_SPARC_TLS_DTPOFF64
BFD_RELOC_SPARC_TLS_TPOFF32
BFD_RELOC_SPARC_TLS_TPOFF64
 SPARC TLS relocations

BFD_RELOC_SPU_IMM7
BFD_RELOC_SPU_IMM8
BFD_RELOC_SPU_IMM10
BFD_RELOC_SPU_IMM10W
BFD_RELOC_SPU_IMM16
BFD_RELOC_SPU_IMM16W
BFD_RELOC_SPU_IMM18
BFD_RELOC_SPU_PCREL9a
BFD_RELOC_SPU_PCREL9b
BFD_RELOC_SPU_PCREL16

BFD_RELOC_SPU_LO16
 BFD_RELOC_SPU_HI16
 BFD_RELOC_SPU_PPU32
 BFD_RELOC_SPU_PPU64
 BFD_RELOC_SPU_ADD_PIC
 SPU Relocations.

BFD_RELOC_ALPHA_GPDISP_HI16

Alpha ECOFF and ELF relocations. Some of these treat the symbol or "addend" in some special way. For GPDISP_HI16 ("gpdisp") relocations, the symbol is ignored when writing; when reading, it will be the absolute section symbol. The addend is the displacement in bytes of the "lda" instruction from the "ldah" instruction (which is at the address of this reloc).

BFD_RELOC_ALPHA_GPDISP_LO16

For GPDISP_LO16 ("ignore") relocations, the symbol is handled as with GPDISP_HI16 relocs. The addend is ignored when writing the relocations out, and is filled in with the file's GP value on reading, for convenience.

BFD_RELOC_ALPHA_GPDISP

The ELF GPDISP relocation is exactly the same as the GPDISP_HI16 relocation except that there is no accompanying GPDISP_LO16 relocation.

BFD_RELOC_ALPHA_LITERAL

BFD_RELOC_ALPHA_ELF_LITERAL

BFD_RELOC_ALPHA_LITUSE

The Alpha LITERAL/LITUSE relocs are produced by a symbol reference; the assembler turns it into a LDQ instruction to load the address of the symbol, and then fills in a register in the real instruction.

The LITERAL reloc, at the LDQ instruction, refers to the .lita section symbol. The addend is ignored when writing, but is filled in with the file's GP value on reading, for convenience, as with the GPDISP_LO16 reloc.

The ELF_LITERAL reloc is somewhere between 16_GOTOFF and GPDISP_LO16. It should refer to the symbol to be referenced, as with 16_GOTOFF, but it generates output not based on the position within the .got section, but relative to the GP value chosen for the file during the final link stage.

The LITUSE reloc, on the instruction using the loaded address, gives information to the linker that it might be able to use to optimize away some literal section references. The symbol is ignored (read as the absolute section symbol), and the "addend" indicates the type of instruction using the register: 1 - "memory" fmt insn 2 - byte-manipulation (byte offset reg) 3 - jsr (target of branch)

BFD_RELOC_ALPHA_HINT

The HINT relocation indicates a value that should be filled into the "hint" field of a jmp/jsr/ret instruction, for possible branch-prediction logic which may be provided on some processors.

BFD_RELOC_ALPHA_LINKAGE

The LINKAGE relocation outputs a linkage pair in the object file, which is filled by the linker.

BFD_RELOC_ALPHA_CODEADDR

The CODEADDR relocation outputs a STO_CA in the object file, which is filled by the linker.

BFD_RELOC_ALPHA_GPREL_HI16**BFD_RELOC_ALPHA_GPREL_LO16**

The GPREL_HI/LO relocations together form a 32-bit offset from the GP register.

BFD_RELOC_ALPHA_BRSGP

Like BFD_RELOC_23_PCREL_S2, except that the source and target must share a common GP, and the target address is adjusted for STO_ALPHA_STD_GPLOAD.

BFD_RELOC_ALPHA_NOP

The NOP relocation outputs a NOP if the longword displacement between two procedure entry points is $< 2^{21}$.

BFD_RELOC_ALPHA_BSR

The BSR relocation outputs a BSR if the longword displacement between two procedure entry points is $< 2^{21}$.

BFD_RELOC_ALPHA_LDA

The LDA relocation outputs a LDA if the longword displacement between two procedure entry points is $< 2^{16}$.

BFD_RELOC_ALPHA_BOH

The BOH relocation outputs a BSR if the longword displacement between two procedure entry points is $< 2^{21}$, or else a hint.

BFD_RELOC_ALPHA_TLSGD**BFD_RELOC_ALPHA_TLSDM****BFD_RELOC_ALPHA_DTPMOD64****BFD_RELOC_ALPHA_GOTDTPREL16****BFD_RELOC_ALPHA_DTPREL64****BFD_RELOC_ALPHA_DTPREL_HI16****BFD_RELOC_ALPHA_DTPREL_LO16****BFD_RELOC_ALPHA_DTPREL16****BFD_RELOC_ALPHA_GOTTPREL16****BFD_RELOC_ALPHA_TPREL64****BFD_RELOC_ALPHA_TPREL_HI16****BFD_RELOC_ALPHA_TPREL_LO16****BFD_RELOC_ALPHA_TPREL16**

Alpha thread-local storage relocations.

BFD_RELOC_MIPS_JMP**BFD_RELOC_MICROMIPS_JMP**

The MIPS jump instruction.

BFD_RELOC_MIPS16_JMP

The MIPS16 jump instruction.

BFD_RELOC_MIPS16_GPREL

MIPS16 GP relative reloc.

BFD_RELOC_HI16

High 16 bits of 32-bit value; simple reloc.

BFD_RELOC_HI16_S

High 16 bits of 32-bit value but the low 16 bits will be sign extended and added to form the final result. If the low 16 bits form a negative number, we need to add one to the high value to compensate for the borrow when the low bits are added.

BFD_RELOC_LO16

Low 16 bits.

BFD_RELOC_HI16_PCREL

High 16 bits of 32-bit pc-relative value

BFD_RELOC_HI16_S_PCREL

High 16 bits of 32-bit pc-relative value, adjusted

BFD_RELOC_LO16_PCREL

Low 16 bits of pc-relative value

BFD_RELOC_MIPS16_GOT16

BFD_RELOC_MIPS16_CALL16

Equivalent of BFD_RELOC_MIPS_*, but with the MIPS16 layout of 16-bit immediate fields

BFD_RELOC_MIPS16_HI16

MIPS16 high 16 bits of 32-bit value.

BFD_RELOC_MIPS16_HI16_S

MIPS16 high 16 bits of 32-bit value but the low 16 bits will be sign extended and added to form the final result. If the low 16 bits form a negative number, we need to add one to the high value to compensate for the borrow when the low bits are added.

BFD_RELOC_MIPS16_LO16

MIPS16 low 16 bits.

BFD_RELOC_MIPS16_TLS_GD

BFD_RELOC_MIPS16_TLS_LDM

BFD_RELOC_MIPS16_TLS_DTPREL_HI16

BFD_RELOC_MIPS16_TLS_DTPREL_LO16

BFD_RELOC_MIPS16_TLS_GOTTPREL

BFD_RELOC_MIPS16_TLS_TPREL_HI16

BFD_RELOC_MIPS16_TLS_TPREL_LO16

MIPS16 TLS relocations

BFD_RELOC_MIPS_LITERAL
BFD_RELOC_MICROMIPS_LITERAL
Relocation against a MIPS literal section.

BFD_RELOC_MICROMIPS_7_PCREL_S1
BFD_RELOC_MICROMIPS_10_PCREL_S1
BFD_RELOC_MICROMIPS_16_PCREL_S1
microMIPS PC-relative relocations.

BFD_RELOC_MIPS16_16_PCREL_S1
MIPS16 PC-relative relocation.

BFD_RELOC_MIPS_21_PCREL_S2
BFD_RELOC_MIPS_26_PCREL_S2
BFD_RELOC_MIPS_18_PCREL_S3
BFD_RELOC_MIPS_19_PCREL_S2
MIPS PC-relative relocations.

BFD_RELOC_MICROMIPS_GPREL16
BFD_RELOC_MICROMIPS_HI16
BFD_RELOC_MICROMIPS_HI16_S
BFD_RELOC_MICROMIPS_LO16
microMIPS versions of generic BFD relocs.

BFD_RELOC_MIPS_GOT16
BFD_RELOC_MICROMIPS_GOT16
BFD_RELOC_MIPS_CALL16
BFD_RELOC_MICROMIPS_CALL16
BFD_RELOC_MIPS_GOT_HI16
BFD_RELOC_MICROMIPS_GOT_HI16
BFD_RELOC_MIPS_GOT_LO16
BFD_RELOC_MICROMIPS_GOT_LO16
BFD_RELOC_MIPS_CALL_HI16
BFD_RELOC_MICROMIPS_CALL_HI16
BFD_RELOC_MIPS_CALL_LO16
BFD_RELOC_MICROMIPS_CALL_LO16
BFD_RELOC_MIPS_SUB
BFD_RELOC_MICROMIPS_SUB
BFD_RELOC_MIPS_GOT_PAGE
BFD_RELOC_MICROMIPS_GOT_PAGE
BFD_RELOC_MIPS_GOT_OFST
BFD_RELOC_MICROMIPS_GOT_OFST
BFD_RELOC_MIPS_GOT_DISP
BFD_RELOC_MICROMIPS_GOT_DISP
BFD_RELOC_MIPS_SHIFT5
BFD_RELOC_MIPS_SHIFT6
BFD_RELOC_MIPS_INSERT_A
BFD_RELOC_MIPS_INSERT_B
BFD_RELOC_MIPS_DELETE

BFD_RELOC_MIPS_HIGHEST
BFD_RELOC_MICROMIPS_HIGHEST
BFD_RELOC_MIPS_HIGHER
BFD_RELOC_MICROMIPS_HIGHER
BFD_RELOC_MIPS_SCN_DISP
BFD_RELOC_MICROMIPS_SCN_DISP
BFD_RELOC_MIPS_16
BFD_RELOC_MIPS_RELGOT
BFD_RELOC_MIPS_JALR
BFD_RELOC_MICROMIPS_JALR
BFD_RELOC_MIPS_TLS_DTPMOD32
BFD_RELOC_MIPS_TLS_DTPREL32
BFD_RELOC_MIPS_TLS_DTPMOD64
BFD_RELOC_MIPS_TLS_DTPREL64
BFD_RELOC_MIPS_TLS_GD
BFD_RELOC_MICROMIPS_TLS_GD
BFD_RELOC_MIPS_TLS_LDM
BFD_RELOC_MICROMIPS_TLS_LDM
BFD_RELOC_MIPS_TLS_DTPREL_HI16
BFD_RELOC_MICROMIPS_TLS_DTPREL_HI16
BFD_RELOC_MIPS_TLS_DTPREL_LO16
BFD_RELOC_MICROMIPS_TLS_DTPREL_LO16
BFD_RELOC_MIPS_TLS_GOTTPREL
BFD_RELOC_MICROMIPS_TLS_GOTTPREL
BFD_RELOC_MIPS_TLS_TPREL32
BFD_RELOC_MIPS_TLS_TPREL64
BFD_RELOC_MIPS_TLS_TPREL_HI16
BFD_RELOC_MICROMIPS_TLS_TPREL_HI16
BFD_RELOC_MIPS_TLS_TPREL_LO16
BFD_RELOC_MICROMIPS_TLS_TPREL_LO16
BFD_RELOC_MIPS_EH

MIPS ELF relocations.

BFD_RELOC_MIPS_COPY
BFD_RELOC_MIPS_JUMP_SLOT

MIPS ELF relocations (VxWorks and PLT extensions).

BFD_RELOC_MOXIE_10_PCREL

Moxie ELF relocations.

BFD_RELOC_FT32_10
BFD_RELOC_FT32_20
BFD_RELOC_FT32_17
BFD_RELOC_FT32_18
BFD_RELOC_FT32_RELAX
BFD_RELOC_FT32_SCO
BFD_RELOC_FT32_SC1
BFD_RELOC_FT32_15

BFD_RELOC_FT32_DIFF32
FT32 ELF relocations.

BFD_RELOC_FRV_LABEL16
BFD_RELOC_FRV_LABEL24
BFD_RELOC_FRV_LO16
BFD_RELOC_FRV_HI16
BFD_RELOC_FRV_GPREL12
BFD_RELOC_FRV_GPRELU12
BFD_RELOC_FRV_GPREL32
BFD_RELOC_FRV_GPRELHI
BFD_RELOC_FRV_GPRELLO
BFD_RELOC_FRV_GOT12
BFD_RELOC_FRV_GOTHI
BFD_RELOC_FRV_GOTLO
BFD_RELOC_FRV_FUNCDESC
BFD_RELOC_FRV_FUNCDESC_GOT12
BFD_RELOC_FRV_FUNCDESC_GOTHI
BFD_RELOC_FRV_FUNCDESC_GOTLO
BFD_RELOC_FRV_FUNCDESC_VALUE
BFD_RELOC_FRV_FUNCDESC_GOTOFF12
BFD_RELOC_FRV_FUNCDESC_GOTOFFHI
BFD_RELOC_FRV_FUNCDESC_GOTOFFLO
BFD_RELOC_FRV_GOTOFF12
BFD_RELOC_FRV_GOTOFFHI
BFD_RELOC_FRV_GOTOFFLO
BFD_RELOC_FRV_GETTLSOFF
BFD_RELOC_FRV_TLSDESC_VALUE
BFD_RELOC_FRV_GOTTLSDESC12
BFD_RELOC_FRV_GOTTLSDESCHI
BFD_RELOC_FRV_GOTTLSDESCLO
BFD_RELOC_FRV_TLMOFF12
BFD_RELOC_FRV_TLMOFFHI
BFD_RELOC_FRV_TLMOFFLO
BFD_RELOC_FRV_GOTTLSOFF12
BFD_RELOC_FRV_GOTTLSOFFHI
BFD_RELOC_FRV_GOTTLSOFFLO
BFD_RELOC_FRV_TLSOFF
BFD_RELOC_FRV_TLSDESC_RELAX
BFD_RELOC_FRV_GETTLSOFF_RELAX
BFD_RELOC_FRV_TLSOFF_RELAX
BFD_RELOC_FRV_TLMOFF
Fujitsu Frv Relocations.

BFD_RELOC_MN10300_GOTOFF24
This is a 24bit GOT-relative reloc for the mn10300.

BFD_RELOC_MN10300_GOT32

This is a 32bit GOT-relative reloc for the mn10300, offset by two bytes in the instruction.

BFD_RELOC_MN10300_GOT24

This is a 24bit GOT-relative reloc for the mn10300, offset by two bytes in the instruction.

BFD_RELOC_MN10300_GOT16

This is a 16bit GOT-relative reloc for the mn10300, offset by two bytes in the instruction.

BFD_RELOC_MN10300_COPY

Copy symbol at runtime.

BFD_RELOC_MN10300_GLOB_DAT

Create GOT entry.

BFD_RELOC_MN10300_JMP_SLOT

Create PLT entry.

BFD_RELOC_MN10300_RELATIVE

Adjust by program base.

BFD_RELOC_MN10300_SYM_DIFF

Together with another reloc targeted at the same location, allows for a value that is the difference of two symbols in the same section.

BFD_RELOC_MN10300_ALIGN

The addend of this reloc is an alignment power that must be honoured at the offset's location, regardless of linker relaxation.

BFD_RELOC_MN10300_TLS_GD**BFD_RELOC_MN10300_TLS_LD****BFD_RELOC_MN10300_TLS_LDO****BFD_RELOC_MN10300_TLS_GOTIE****BFD_RELOC_MN10300_TLS_IE****BFD_RELOC_MN10300_TLS_LE****BFD_RELOC_MN10300_TLS_DTPMOD****BFD_RELOC_MN10300_TLS_DTPOFF****BFD_RELOC_MN10300_TLS_TPOFF**

Various TLS-related relocations.

BFD_RELOC_MN10300_32_PCREL

This is a 32bit precl reloc for the mn10300, offset by two bytes in the instruction.

BFD_RELOC_MN10300_16_PCREL

This is a 16bit precl reloc for the mn10300, offset by two bytes in the instruction.

BFD_RELOC_386_GOT32
BFD_RELOC_386_PLT32
BFD_RELOC_386_COPY
BFD_RELOC_386_GLOB_DAT
BFD_RELOC_386_JUMP_SLOT
BFD_RELOC_386_RELATIVE
BFD_RELOC_386_GOTOFF
BFD_RELOC_386_GOTPC
BFD_RELOC_386_TLS_TPOFF
BFD_RELOC_386_TLS_IE
BFD_RELOC_386_TLS_GOTIE
BFD_RELOC_386_TLS_LE
BFD_RELOC_386_TLS_GD
BFD_RELOC_386_TLS_LDM
BFD_RELOC_386_TLS_LDO_32
BFD_RELOC_386_TLS_IE_32
BFD_RELOC_386_TLS_LE_32
BFD_RELOC_386_TLS_DTPMOD32
BFD_RELOC_386_TLS_DTPOFF32
BFD_RELOC_386_TLS_TPOFF32
BFD_RELOC_386_TLS_GOTDESC
BFD_RELOC_386_TLS_DESC_CALL
BFD_RELOC_386_TLS_DESC
BFD_RELOC_386_IRELATIVE
BFD_RELOC_386_GOT32X
 i386/elf relocations

BFD_RELOC_X86_64_GOT32
BFD_RELOC_X86_64_PLT32
BFD_RELOC_X86_64_COPY
BFD_RELOC_X86_64_GLOB_DAT
BFD_RELOC_X86_64_JUMP_SLOT
BFD_RELOC_X86_64_RELATIVE
BFD_RELOC_X86_64_GOTPCREL
BFD_RELOC_X86_64_32S
BFD_RELOC_X86_64_DTPMOD64
BFD_RELOC_X86_64_DTPOFF64
BFD_RELOC_X86_64_TPOFF64
BFD_RELOC_X86_64_TLSD
BFD_RELOC_X86_64_TLSD
BFD_RELOC_X86_64_DTPOFF32
BFD_RELOC_X86_64_GOTTPOFF
BFD_RELOC_X86_64_TPOFF32
BFD_RELOC_X86_64_GOTOFF64
BFD_RELOC_X86_64_GOTPC32
BFD_RELOC_X86_64_GOT64
BFD_RELOC_X86_64_GOTPCREL64

BFD_RELOC_X86_64_GOTPC64
BFD_RELOC_X86_64_GOTPLT64
BFD_RELOC_X86_64_PLTOFF64
BFD_RELOC_X86_64_GOTPC32_TLSDESC
BFD_RELOC_X86_64_TLSDESC_CALL
BFD_RELOC_X86_64_TLSDESC
BFD_RELOC_X86_64_IRELATIVE
BFD_RELOC_X86_64_PC32_BND
BFD_RELOC_X86_64_PLT32_BND
BFD_RELOC_X86_64_GOTPCRELX
BFD_RELOC_X86_64_REX_GOTPCRELX
x86-64/elf relocations

BFD_RELOC_NS32K_IMM_8
BFD_RELOC_NS32K_IMM_16
BFD_RELOC_NS32K_IMM_32
BFD_RELOC_NS32K_IMM_8_PCREL
BFD_RELOC_NS32K_IMM_16_PCREL
BFD_RELOC_NS32K_IMM_32_PCREL
BFD_RELOC_NS32K_DISP_8
BFD_RELOC_NS32K_DISP_16
BFD_RELOC_NS32K_DISP_32
BFD_RELOC_NS32K_DISP_8_PCREL
BFD_RELOC_NS32K_DISP_16_PCREL
BFD_RELOC_NS32K_DISP_32_PCREL
ns32k relocations

BFD_RELOC_PDP11_DISP_8_PCREL
BFD_RELOC_PDP11_DISP_6_PCREL
PDP11 relocations

BFD_RELOC_PJ_CODE_HI16
BFD_RELOC_PJ_CODE_LO16
BFD_RELOC_PJ_CODE_DIR16
BFD_RELOC_PJ_CODE_DIR32
BFD_RELOC_PJ_CODE_REL16
BFD_RELOC_PJ_CODE_REL32

Picojava relocs. Not all of these appear in object files.

BFD_RELOC_PPC_B26
BFD_RELOC_PPC_BA26
BFD_RELOC_PPC_TOC16
BFD_RELOC_PPC_TOC16_LO
BFD_RELOC_PPC_TOC16_HI
BFD_RELOC_PPC_B16
BFD_RELOC_PPC_B16_BRTAKEN
BFD_RELOC_PPC_B16_BRNTAKEN
BFD_RELOC_PPC_BA16

BFD_RELOC_PPC_BA16_BRTAKEN
BFD_RELOC_PPC_BA16_BRNTAKEN
BFD_RELOC_PPC_COPY
BFD_RELOC_PPC_GLOB_DAT
BFD_RELOC_PPC_JMP_SLOT
BFD_RELOC_PPC_RELATIVE
BFD_RELOC_PPC_LOCAL24PC
BFD_RELOC_PPC_EMB_NADDR32
BFD_RELOC_PPC_EMB_NADDR16
BFD_RELOC_PPC_EMB_NADDR16_LO
BFD_RELOC_PPC_EMB_NADDR16_HI
BFD_RELOC_PPC_EMB_NADDR16_HA
BFD_RELOC_PPC_EMB_SDAI16
BFD_RELOC_PPC_EMB_SDA2I16
BFD_RELOC_PPC_EMB_SDA2REL
BFD_RELOC_PPC_EMB_SDA21
BFD_RELOC_PPC_EMB_MRKREF
BFD_RELOC_PPC_EMB_RELSEC16
BFD_RELOC_PPC_EMB_RELST_LO
BFD_RELOC_PPC_EMB_RELST_HI
BFD_RELOC_PPC_EMB_RELST_HA
BFD_RELOC_PPC_EMB_BIT_FLD
BFD_RELOC_PPC_EMB_RELSDA
BFD_RELOC_PPC_VLE_REL8
BFD_RELOC_PPC_VLE_REL15
BFD_RELOC_PPC_VLE_REL24
BFD_RELOC_PPC_VLE_LO16A
BFD_RELOC_PPC_VLE_LO16D
BFD_RELOC_PPC_VLE_HI16A
BFD_RELOC_PPC_VLE_HI16D
BFD_RELOC_PPC_VLE_HA16A
BFD_RELOC_PPC_VLE_HA16D
BFD_RELOC_PPC_VLE_SDA21
BFD_RELOC_PPC_VLE_SDA21_LO
BFD_RELOC_PPC_VLE_SDAREL_LO16A
BFD_RELOC_PPC_VLE_SDAREL_LO16D
BFD_RELOC_PPC_VLE_SDAREL_HI16A
BFD_RELOC_PPC_VLE_SDAREL_HI16D
BFD_RELOC_PPC_VLE_SDAREL_HA16A
BFD_RELOC_PPC_VLE_SDAREL_HA16D
BFD_RELOC_PPC_16DX_HA
BFD_RELOC_PPC_REL16DX_HA
BFD_RELOC_PPC_NEG
BFD_RELOC_PPC64_HIGHER
BFD_RELOC_PPC64_HIGHER_S
BFD_RELOC_PPC64_HIGHEST
BFD_RELOC_PPC64_HIGHEST_S

BFD_RELOC_PPC64_TOC16_LO
BFD_RELOC_PPC64_TOC16_HI
BFD_RELOC_PPC64_TOC16_HA
BFD_RELOC_PPC64_TOC
BFD_RELOC_PPC64_PLTGOT16
BFD_RELOC_PPC64_PLTGOT16_LO
BFD_RELOC_PPC64_PLTGOT16_HI
BFD_RELOC_PPC64_PLTGOT16_HA
BFD_RELOC_PPC64_ADDR16_DS
BFD_RELOC_PPC64_ADDR16_LO_DS
BFD_RELOC_PPC64_GOT16_DS
BFD_RELOC_PPC64_GOT16_LO_DS
BFD_RELOC_PPC64_PLT16_LO_DS
BFD_RELOC_PPC64_SECTOFF_DS
BFD_RELOC_PPC64_SECTOFF_LO_DS
BFD_RELOC_PPC64_TOC16_DS
BFD_RELOC_PPC64_TOC16_LO_DS
BFD_RELOC_PPC64_PLTGOT16_DS
BFD_RELOC_PPC64_PLTGOT16_LO_DS
BFD_RELOC_PPC64_ADDR16_HIGH
BFD_RELOC_PPC64_ADDR16_HIGHA
BFD_RELOC_PPC64_REL16_HIGH
BFD_RELOC_PPC64_REL16_HIGHA
BFD_RELOC_PPC64_REL16_HIGHER
BFD_RELOC_PPC64_REL16_HIGHERA
BFD_RELOC_PPC64_REL16_HIGHEST
BFD_RELOC_PPC64_REL16_HIGHESTA
BFD_RELOC_PPC64_ADDR64_LOCAL
BFD_RELOC_PPC64_ENTRY
BFD_RELOC_PPC64_REL24_NOTOC
BFD_RELOC_PPC64_REL24_P9NOTOC
BFD_RELOC_PPC64_D34
BFD_RELOC_PPC64_D34_LO
BFD_RELOC_PPC64_D34_HI30
BFD_RELOC_PPC64_D34_HA30
BFD_RELOC_PPC64_PCREL34
BFD_RELOC_PPC64_GOT_PCREL34
BFD_RELOC_PPC64_PLT_PCREL34
BFD_RELOC_PPC64_ADDR16_HIGHER34
BFD_RELOC_PPC64_ADDR16_HIGHERA34
BFD_RELOC_PPC64_ADDR16_HIGHEST34
BFD_RELOC_PPC64_ADDR16_HIGHESTA34
BFD_RELOC_PPC64_REL16_HIGHER34
BFD_RELOC_PPC64_REL16_HIGHERA34
BFD_RELOC_PPC64_REL16_HIGHEST34
BFD_RELOC_PPC64_REL16_HIGHESTA34
BFD_RELOC_PPC64_D28

BFD_RELOC_PPC64_PCREL28

Power(rs6000) and PowerPC relocations.

BFD_RELOC_PPC_TLS
BFD_RELOC_PPC_TLSGD
BFD_RELOC_PPC_TLSLD
BFD_RELOC_PPC_TLSLE
BFD_RELOC_PPC_TLSIE
BFD_RELOC_PPC_TLSM
BFD_RELOC_PPC_TLSQL
BFD_RELOC_PPC_DTPMOD
BFD_RELOC_PPC_TPREL16
BFD_RELOC_PPC_TPREL16_LO
BFD_RELOC_PPC_TPREL16_HI
BFD_RELOC_PPC_TPREL16_HA
BFD_RELOC_PPC_TPREL
BFD_RELOC_PPC_DTPREL16
BFD_RELOC_PPC_DTPREL16_LO
BFD_RELOC_PPC_DTPREL16_HI
BFD_RELOC_PPC_DTPREL16_HA
BFD_RELOC_PPC_DTPREL
BFD_RELOC_PPC_GOT_TLSGD16
BFD_RELOC_PPC_GOT_TLSGD16_LO
BFD_RELOC_PPC_GOT_TLSGD16_HI
BFD_RELOC_PPC_GOT_TLSGD16_HA
BFD_RELOC_PPC_GOT_TLSLD16
BFD_RELOC_PPC_GOT_TLSLD16_LO
BFD_RELOC_PPC_GOT_TLSLD16_HI
BFD_RELOC_PPC_GOT_TLSLD16_HA
BFD_RELOC_PPC_GOT_TPREL16
BFD_RELOC_PPC_GOT_TPREL16_LO
BFD_RELOC_PPC_GOT_TPREL16_HI
BFD_RELOC_PPC_GOT_TPREL16_HA
BFD_RELOC_PPC_GOT_DTPREL16
BFD_RELOC_PPC_GOT_DTPREL16_LO
BFD_RELOC_PPC_GOT_DTPREL16_HI
BFD_RELOC_PPC_GOT_DTPREL16_HA
BFD_RELOC_PPC64_TLSGD
BFD_RELOC_PPC64_TLSLD
BFD_RELOC_PPC64_TLSLE
BFD_RELOC_PPC64_TLSIE
BFD_RELOC_PPC64_TLSM
BFD_RELOC_PPC64_TLSQL
BFD_RELOC_PPC64_TPREL16_DS
BFD_RELOC_PPC64_TPREL16_LO_DS
BFD_RELOC_PPC64_TPREL16_HIGH
BFD_RELOC_PPC64_TPREL16_HIGHA

BFD_RELOC_PPC64_TPREL16_HIGHER
 BFD_RELOC_PPC64_TPREL16_HIGHERA
 BFD_RELOC_PPC64_TPREL16_HIGHEST
 BFD_RELOC_PPC64_TPREL16_HIGHESTA
 BFD_RELOC_PPC64_DTPREL16_DS
 BFD_RELOC_PPC64_DTPREL16_LO_DS
 BFD_RELOC_PPC64_DTPREL16_HIGH
 BFD_RELOC_PPC64_DTPREL16_HIGHA
 BFD_RELOC_PPC64_DTPREL16_HIGHER
 BFD_RELOC_PPC64_DTPREL16_HIGHERA
 BFD_RELOC_PPC64_DTPREL16_HIGHEST
 BFD_RELOC_PPC64_DTPREL16_HIGHESTA
 BFD_RELOC_PPC64_TPREL34
 BFD_RELOC_PPC64_DTPREL34
 BFD_RELOC_PPC64_GOT_TLSDG_PCREL34
 BFD_RELOC_PPC64_GOT_TLSD_PCREL34
 BFD_RELOC_PPC64_GOT_TPREL_PCREL34
 BFD_RELOC_PPC64_GOT_DTPREL_PCREL34
 BFD_RELOC_PPC64_TLS_PCREL

PowerPC and PowerPC64 thread-local storage relocations.

BFD_RELOC_I370_D12
 IBM 370/390 relocations

BFD_RELOC_CTOR

The type of reloc used to build a constructor table - at the moment probably a 32 bit wide absolute relocation, but the target can choose. It generally does map to one of the other relocation types.

BFD_RELOC_ARM_PCREL_BRANCH

ARM 26 bit pc-relative branch. The lowest two bits must be zero and are not stored in the instruction.

BFD_RELOC_ARM_PCREL_BLX

ARM 26 bit pc-relative branch. The lowest bit must be zero and is not stored in the instruction. The 2nd lowest bit comes from a 1 bit field in the instruction.

BFD_RELOC_THUMB_PCREL_BLX

Thumb 22 bit pc-relative branch. The lowest bit must be zero and is not stored in the instruction. The 2nd lowest bit comes from a 1 bit field in the instruction.

BFD_RELOC_ARM_PCREL_CALL

ARM 26-bit pc-relative branch for an unconditional BL or BLX instruction.

BFD_RELOC_ARM_PCREL_JUMP

ARM 26-bit pc-relative branch for B or conditional BL instruction.

BFD_RELOC_THUMB_PCREL_BRANCH5

ARM 5-bit pc-relative branch for Branch Future instructions.

BFD_RELOC_THUMB_PCREL_BFCSEL

ARM 6-bit pc-relative branch for BFCSEL instruction.

BFD_RELOC_ARM_THUMB_BF17

ARM 17-bit pc-relative branch for Branch Future instructions.

BFD_RELOC_ARM_THUMB_BF13

ARM 13-bit pc-relative branch for BFCSEL instruction.

BFD_RELOC_ARM_THUMB_BF19

ARM 19-bit pc-relative branch for Branch Future Link instruction.

BFD_RELOC_ARM_THUMB_LOOP12

ARM 12-bit pc-relative branch for Low Overhead Loop instructions.

BFD_RELOC_THUMB_PCREL_BRANCH7**BFD_RELOC_THUMB_PCREL_BRANCH9****BFD_RELOC_THUMB_PCREL_BRANCH12****BFD_RELOC_THUMB_PCREL_BRANCH20****BFD_RELOC_THUMB_PCREL_BRANCH23****BFD_RELOC_THUMB_PCREL_BRANCH25**

Thumb 7-, 9-, 12-, 20-, 23-, and 25-bit pc-relative branches. The lowest bit must be zero and is not stored in the instruction. Note that the corresponding ELF `R_ARM_THM_JUMPnn` constant has an "nn" one smaller in all cases. Note further that `BRANCH23` corresponds to `R_ARM_THM_CALL`.

BFD_RELOC_ARM_OFFSET_IMM

12-bit immediate offset, used in ARM-format `ldr` and `str` instructions.

BFD_RELOC_ARM_THUMB_OFFSET

5-bit immediate offset, used in Thumb-format `ldr` and `str` instructions.

BFD_RELOC_ARM_TARGET1

Pc-relative or absolute relocation depending on target. Used for entries in `.init_array` sections.

BFD_RELOC_ARM_ROSEGREL32

Read-only segment base relative address.

BFD_RELOC_ARM_SBREL32

Data segment base relative address.

BFD_RELOC_ARM_TARGET2

This reloc is used for references to RTTI data from exception handling tables. The actual definition depends on the target. It may be a pc-relative or some form of GOT-indirect relocation.

BFD_RELOC_ARM_PREL31

31-bit PC relative address.

BFD_RELOC_ARM_MOVW
 BFD_RELOC_ARM_MOVT
 BFD_RELOC_ARM_MOVW_PCREL
 BFD_RELOC_ARM_MOVT_PCREL
 BFD_RELOC_ARM_THUMB_MOVW
 BFD_RELOC_ARM_THUMB_MOVT
 BFD_RELOC_ARM_THUMB_MOVW_PCREL
 BFD_RELOC_ARM_THUMB_MOVT_PCREL

Low and High halfword relocations for MOVW and MOVT instructions.

BFD_RELOC_ARM_GOTFUNCDESC
 BFD_RELOC_ARM_GOTOFFFUNCDESC
 BFD_RELOC_ARM_FUNCDESC
 BFD_RELOC_ARM_FUNCDESC_VALUE
 BFD_RELOC_ARM_TLS_GD32_FDPIC
 BFD_RELOC_ARM_TLS_LDM32_FDPIC
 BFD_RELOC_ARM_TLS_IE32_FDPIC

ARM FDPIC specific relocations.

BFD_RELOC_ARM_JUMP_SLOT
 BFD_RELOC_ARM_GLOB_DAT
 BFD_RELOC_ARM_GOT32
 BFD_RELOC_ARM_PLT32
 BFD_RELOC_ARM_RELATIVE
 BFD_RELOC_ARM_GOTOFF
 BFD_RELOC_ARM_GOTPC
 BFD_RELOC_ARM_GOT_PREL

Relocations for setting up GOTs and PLTs for shared libraries.

BFD_RELOC_ARM_TLS_GD32
 BFD_RELOC_ARM_TLS_LD032
 BFD_RELOC_ARM_TLS_LDM32
 BFD_RELOC_ARM_TLS_DTPOFF32
 BFD_RELOC_ARM_TLS_DTPMOD32
 BFD_RELOC_ARM_TLS_TPOFF32
 BFD_RELOC_ARM_TLS_IE32
 BFD_RELOC_ARM_TLS_LE32
 BFD_RELOC_ARM_TLS_GOTDESC
 BFD_RELOC_ARM_TLS_CALL
 BFD_RELOC_ARM_THM_TLS_CALL
 BFD_RELOC_ARM_TLS_DESCSEQ
 BFD_RELOC_ARM_THM_TLS_DESCSEQ
 BFD_RELOC_ARM_TLS_DESC

ARM thread-local storage relocations.

BFD_RELOC_ARM_ALU_PC_GO_NC
 BFD_RELOC_ARM_ALU_PC_GO
 BFD_RELOC_ARM_ALU_PC_G1_NC

BFD_RELOC_ARM_ALU_PC_G1
BFD_RELOC_ARM_ALU_PC_G2
BFD_RELOC_ARM_LDR_PC_G0
BFD_RELOC_ARM_LDR_PC_G1
BFD_RELOC_ARM_LDR_PC_G2
BFD_RELOC_ARM_LDRS_PC_G0
BFD_RELOC_ARM_LDRS_PC_G1
BFD_RELOC_ARM_LDRS_PC_G2
BFD_RELOC_ARM_LDC_PC_G0
BFD_RELOC_ARM_LDC_PC_G1
BFD_RELOC_ARM_LDC_PC_G2
BFD_RELOC_ARM_ALU_SB_G0_NC
BFD_RELOC_ARM_ALU_SB_G0
BFD_RELOC_ARM_ALU_SB_G1_NC
BFD_RELOC_ARM_ALU_SB_G1
BFD_RELOC_ARM_ALU_SB_G2
BFD_RELOC_ARM_LDR_SB_G0
BFD_RELOC_ARM_LDR_SB_G1
BFD_RELOC_ARM_LDR_SB_G2
BFD_RELOC_ARM_LDRS_SB_G0
BFD_RELOC_ARM_LDRS_SB_G1
BFD_RELOC_ARM_LDRS_SB_G2
BFD_RELOC_ARM_LDC_SB_G0
BFD_RELOC_ARM_LDC_SB_G1
BFD_RELOC_ARM_LDC_SB_G2

ARM group relocations.

BFD_RELOC_ARM_V4BX

Annotation of BX instructions.

BFD_RELOC_ARM_IRELATIVE

ARM support for STT_GNU_IFUNC.

BFD_RELOC_ARM_THUMB_ALU_ABS_G0_NC
BFD_RELOC_ARM_THUMB_ALU_ABS_G1_NC
BFD_RELOC_ARM_THUMB_ALU_ABS_G2_NC
BFD_RELOC_ARM_THUMB_ALU_ABS_G3_NC

Thumb1 relocations to support execute-only code.

BFD_RELOC_ARM_IMMEDIATE
BFD_RELOC_ARM_ADRL_IMMEDIATE
BFD_RELOC_ARM_T32_IMMEDIATE
BFD_RELOC_ARM_T32_ADD_IMM
BFD_RELOC_ARM_T32_IMM12
BFD_RELOC_ARM_T32_ADD_PC12
BFD_RELOC_ARM_SHIFT_IMM
BFD_RELOC_ARM_SMC
BFD_RELOC_ARM_HVC

BFD_RELOC_ARM_SWI
BFD_RELOC_ARM_MULTI
BFD_RELOC_ARM_CP_OFF_IMM
BFD_RELOC_ARM_CP_OFF_IMM_S2
BFD_RELOC_ARM_T32_CP_OFF_IMM
BFD_RELOC_ARM_T32_CP_OFF_IMM_S2
BFD_RELOC_ARM_T32_VLDR_VSTR_OFF_IMM
BFD_RELOC_ARM_ADR_IMM
BFD_RELOC_ARM_LDR_IMM
BFD_RELOC_ARM_LITERAL
BFD_RELOC_ARM_IN_POOL
BFD_RELOC_ARM_OFFSET_IMM8
BFD_RELOC_ARM_T32_OFFSET_U8
BFD_RELOC_ARM_T32_OFFSET_IMM
BFD_RELOC_ARM_HWLITERAL
BFD_RELOC_ARM_THUMB_ADD
BFD_RELOC_ARM_THUMB_IMM
BFD_RELOC_ARM_THUMB_SHIFT

These relocs are only used within the ARM assembler. They are not (at present) written to any object files.

BFD_RELOC_SH_PCDISP8BY2
BFD_RELOC_SH_PCDISP12BY2
BFD_RELOC_SH_IMM3
BFD_RELOC_SH_IMM3U
BFD_RELOC_SH_DISP12
BFD_RELOC_SH_DISP12BY2
BFD_RELOC_SH_DISP12BY4
BFD_RELOC_SH_DISP12BY8
BFD_RELOC_SH_DISP20
BFD_RELOC_SH_DISP20BY8
BFD_RELOC_SH_IMM4
BFD_RELOC_SH_IMM4BY2
BFD_RELOC_SH_IMM4BY4
BFD_RELOC_SH_IMM8
BFD_RELOC_SH_IMM8BY2
BFD_RELOC_SH_IMM8BY4
BFD_RELOC_SH_PCRELIMM8BY2
BFD_RELOC_SH_PCRELIMM8BY4
BFD_RELOC_SH_SWITCH16
BFD_RELOC_SH_SWITCH32
BFD_RELOC_SH_USES
BFD_RELOC_SH_COUNT
BFD_RELOC_SH_ALIGN
BFD_RELOC_SH_CODE
BFD_RELOC_SH_DATA
BFD_RELOC_SH_LABEL

BFD_RELOC_SH_LOOP_START
BFD_RELOC_SH_LOOP_END
BFD_RELOC_SH_COPY
BFD_RELOC_SH_GLOB_DAT
BFD_RELOC_SH_JMP_SLOT
BFD_RELOC_SH_RELATIVE
BFD_RELOC_SH_GOTPC
BFD_RELOC_SH_GOT_LOW16
BFD_RELOC_SH_GOT_MEDLOW16
BFD_RELOC_SH_GOT_MEDHI16
BFD_RELOC_SH_GOT_HI16
BFD_RELOC_SH_GOTPLT_LOW16
BFD_RELOC_SH_GOTPLT_MEDLOW16
BFD_RELOC_SH_GOTPLT_MEDHI16
BFD_RELOC_SH_GOTPLT_HI16
BFD_RELOC_SH_PLT_LOW16
BFD_RELOC_SH_PLT_MEDLOW16
BFD_RELOC_SH_PLT_MEDHI16
BFD_RELOC_SH_PLT_HI16
BFD_RELOC_SH_GOTOFF_LOW16
BFD_RELOC_SH_GOTOFF_MEDLOW16
BFD_RELOC_SH_GOTOFF_MEDHI16
BFD_RELOC_SH_GOTOFF_HI16
BFD_RELOC_SH_GOTPC_LOW16
BFD_RELOC_SH_GOTPC_MEDLOW16
BFD_RELOC_SH_GOTPC_MEDHI16
BFD_RELOC_SH_GOTPC_HI16
BFD_RELOC_SH_COPY64
BFD_RELOC_SH_GLOB_DAT64
BFD_RELOC_SH_JMP_SLOT64
BFD_RELOC_SH_RELATIVE64
BFD_RELOC_SH_GOT10BY4
BFD_RELOC_SH_GOT10BY8
BFD_RELOC_SH_GOTPLT10BY4
BFD_RELOC_SH_GOTPLT10BY8
BFD_RELOC_SH_GOTPLT32
BFD_RELOC_SH_SHMEDIA_CODE
BFD_RELOC_SH_IMMU5
BFD_RELOC_SH_IMMS6
BFD_RELOC_SH_IMMS6BY32
BFD_RELOC_SH_IMMU6
BFD_RELOC_SH_IMMS10
BFD_RELOC_SH_IMMS10BY2
BFD_RELOC_SH_IMMS10BY4
BFD_RELOC_SH_IMMS10BY8
BFD_RELOC_SH_IMMS16
BFD_RELOC_SH_IMMU16

BFD_RELOC_SH_IMM_LOW16
BFD_RELOC_SH_IMM_LOW16_PCREL
BFD_RELOC_SH_IMM_MEDLOW16
BFD_RELOC_SH_IMM_MEDLOW16_PCREL
BFD_RELOC_SH_IMM_MEDHI16
BFD_RELOC_SH_IMM_MEDHI16_PCREL
BFD_RELOC_SH_IMM_HI16
BFD_RELOC_SH_IMM_HI16_PCREL
BFD_RELOC_SH_PT_16
BFD_RELOC_SH_TLS_GD_32
BFD_RELOC_SH_TLS_LD_32
BFD_RELOC_SH_TLS_LDO_32
BFD_RELOC_SH_TLS_IE_32
BFD_RELOC_SH_TLS_LE_32
BFD_RELOC_SH_TLS_DTPMOD32
BFD_RELOC_SH_TLS_DTPOFF32
BFD_RELOC_SH_TLS_TPOFF32
BFD_RELOC_SH_GOT20
BFD_RELOC_SH_GOTOFF20
BFD_RELOC_SH_GOTFUNCDESC
BFD_RELOC_SH_GOTFUNCDESC20
BFD_RELOC_SH_GOTOFFFUNCDESC
BFD_RELOC_SH_GOTOFFFUNCDESC20
BFD_RELOC_SH_FUNCDESC

Renesas / SuperH SH relocs. Not all of these appear in object files.

BFD_RELOC_ARC_NONE
BFD_RELOC_ARC_8
BFD_RELOC_ARC_16
BFD_RELOC_ARC_24
BFD_RELOC_ARC_32
BFD_RELOC_ARC_N8
BFD_RELOC_ARC_N16
BFD_RELOC_ARC_N24
BFD_RELOC_ARC_N32
BFD_RELOC_ARC_SDA
BFD_RELOC_ARC_SECTOFF
BFD_RELOC_ARC_S21H_PCREL
BFD_RELOC_ARC_S21W_PCREL
BFD_RELOC_ARC_S25H_PCREL
BFD_RELOC_ARC_S25W_PCREL
BFD_RELOC_ARC_SDA32
BFD_RELOC_ARC_SDA_LDST
BFD_RELOC_ARC_SDA_LDST1
BFD_RELOC_ARC_SDA_LDST2
BFD_RELOC_ARC_SDA16_LD
BFD_RELOC_ARC_SDA16_LD1

BFD_RELOC_ARC_SDA16_LD2
BFD_RELOC_ARC_S13_PCREL
BFD_RELOC_ARC_W
BFD_RELOC_ARC_32_ME
BFD_RELOC_ARC_32_ME_S
BFD_RELOC_ARC_N32_ME
BFD_RELOC_ARC_SECTOFF_ME
BFD_RELOC_ARC_SDA32_ME
BFD_RELOC_ARC_W_ME
BFD_RELOC_AC_SECTOFF_U8
BFD_RELOC_AC_SECTOFF_U8_1
BFD_RELOC_AC_SECTOFF_U8_2
BFD_RELOC_AC_SECTOFF_S9
BFD_RELOC_AC_SECTOFF_S9_1
BFD_RELOC_AC_SECTOFF_S9_2
BFD_RELOC_ARC_SECTOFF_ME_1
BFD_RELOC_ARC_SECTOFF_ME_2
BFD_RELOC_ARC_SECTOFF_1
BFD_RELOC_ARC_SECTOFF_2
BFD_RELOC_ARC_SDA_12
BFD_RELOC_ARC_SDA16_ST2
BFD_RELOC_ARC_32_PCREL
BFD_RELOC_ARC_PC32
BFD_RELOC_ARC_GOT32
BFD_RELOC_ARC_GOTPC32
BFD_RELOC_ARC_PLT32
BFD_RELOC_ARC_COPY
BFD_RELOC_ARC_GLOB_DAT
BFD_RELOC_ARC_JMP_SLOT
BFD_RELOC_ARC_RELATIVE
BFD_RELOC_ARC_GOTOFF
BFD_RELOC_ARC_GOTPC
BFD_RELOC_ARC_S21W_PCREL_PLT
BFD_RELOC_ARC_S25H_PCREL_PLT
BFD_RELOC_ARC_TLS_DTPMOD
BFD_RELOC_ARC_TLS_TPOFF
BFD_RELOC_ARC_TLS_GD_GOT
BFD_RELOC_ARC_TLS_GD_LD
BFD_RELOC_ARC_TLS_GD_CALL
BFD_RELOC_ARC_TLS_IE_GOT
BFD_RELOC_ARC_TLS_DTPOFF
BFD_RELOC_ARC_TLS_DTPOFF_S9
BFD_RELOC_ARC_TLS_LE_S9
BFD_RELOC_ARC_TLS_LE_32
BFD_RELOC_ARC_S25W_PCREL_PLT
BFD_RELOC_ARC_S21H_PCREL_PLT
BFD_RELOC_ARC_NPS_CMEM16

BFD_RELOC_ARC_JLI_SECTOFF

ARC relocs.

BFD_RELOC_BFIN_16_IMM

ADI Blackfin 16 bit immediate absolute reloc.

BFD_RELOC_BFIN_16_HIGH

ADI Blackfin 16 bit immediate absolute reloc higher 16 bits.

BFD_RELOC_BFIN_4_PCREL

ADI Blackfin 'a' part of LSETUP.

BFD_RELOC_BFIN_5_PCREL

ADI Blackfin.

BFD_RELOC_BFIN_16_LOW

ADI Blackfin 16 bit immediate absolute reloc lower 16 bits.

BFD_RELOC_BFIN_10_PCREL

ADI Blackfin.

BFD_RELOC_BFIN_11_PCREL

ADI Blackfin 'b' part of LSETUP.

BFD_RELOC_BFIN_12_PCREL_JUMP

ADI Blackfin.

BFD_RELOC_BFIN_12_PCREL_JUMP_S

ADI Blackfin Short jump, pcrel.

BFD_RELOC_BFIN_24_PCREL_CALL_X

ADI Blackfin Call.x not implemented.

BFD_RELOC_BFIN_24_PCREL_JUMP_L

ADI Blackfin Long Jump pcrel.

BFD_RELOC_BFIN_GOT17M4

BFD_RELOC_BFIN_GOTHI

BFD_RELOC_BFIN_GOTLO

BFD_RELOC_BFIN_FUNCDESC

BFD_RELOC_BFIN_FUNCDESC_GOT17M4

BFD_RELOC_BFIN_FUNCDESC_GOTHI

BFD_RELOC_BFIN_FUNCDESC_GOTLO

BFD_RELOC_BFIN_FUNCDESC_VALUE

BFD_RELOC_BFIN_FUNCDESC_GOTOFF17M4

BFD_RELOC_BFIN_FUNCDESC_GOTOFFHI

BFD_RELOC_BFIN_FUNCDESC_GOTOFFLO

BFD_RELOC_BFIN_GOTOFF17M4

BFD_RELOC_BFIN_GOTOFFHI

BFD_RELOC_BFIN_GOTOFFLO

ADI Blackfin FD-PIC relocations.

BFD_RELOC_BFIN_GOT
ADI Blackfin GOT relocation.

BFD_RELOC_BFIN_PLTPC
ADI Blackfin PLTPC relocation.

BFD_ARELOC_BFIN_PUSH
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_CONST
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_ADD
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_SUB
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_MULT
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_DIV
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_MOD
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_LSHIFT
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_RSHIFT
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_AND
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_OR
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_XOR
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_LAND
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_LOR
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_LEN
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_NEG
ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_COMP

ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_PAGE

ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_HWPAGE

ADI Blackfin arithmetic relocation.

BFD_ARELOC_BFIN_ADDR

ADI Blackfin arithmetic relocation.

BFD_RELOC_D10V_10_PCREL_R

Mitsubishi D10V relocs. This is a 10-bit reloc with the right 2 bits assumed to be 0.

BFD_RELOC_D10V_10_PCREL_L

Mitsubishi D10V relocs. This is a 10-bit reloc with the right 2 bits assumed to be 0. This is the same as the previous reloc except it is in the left container, i.e., shifted left 15 bits.

BFD_RELOC_D10V_18

This is an 18-bit reloc with the right 2 bits assumed to be 0.

BFD_RELOC_D10V_18_PCREL

This is an 18-bit reloc with the right 2 bits assumed to be 0.

BFD_RELOC_D30V_6

Mitsubishi D30V relocs. This is a 6-bit absolute reloc.

BFD_RELOC_D30V_9_PCREL

This is a 6-bit pc-relative reloc with the right 3 bits assumed to be 0.

BFD_RELOC_D30V_9_PCREL_R

This is a 6-bit pc-relative reloc with the right 3 bits assumed to be 0. Same as the previous reloc but on the right side of the container.

BFD_RELOC_D30V_15

This is a 12-bit absolute reloc with the right 3 bits assumed to be 0.

BFD_RELOC_D30V_15_PCREL

This is a 12-bit pc-relative reloc with the right 3 bits assumed to be 0.

BFD_RELOC_D30V_15_PCREL_R

This is a 12-bit pc-relative reloc with the right 3 bits assumed to be 0. Same as the previous reloc but on the right side of the container.

BFD_RELOC_D30V_21

This is an 18-bit absolute reloc with the right 3 bits assumed to be 0.

BFD_RELOC_D30V_21_PCREL

This is an 18-bit pc-relative reloc with the right 3 bits assumed to be 0.

BFD_RELOC_D30V_21_PCREL_R

This is an 18-bit pc-relative reloc with the right 3 bits assumed to be 0. Same as the previous reloc but on the right side of the container.

BFD_RELOC_D30V_32

This is a 32-bit absolute reloc.

BFD_RELOC_D30V_32_PCREL

This is a 32-bit pc-relative reloc.

BFD_RELOC_DLX_HI16_S

DLX relocs

BFD_RELOC_DLX_L016

DLX relocs

BFD_RELOC_DLX_JMP26

DLX relocs

BFD_RELOC_M32C_HI8**BFD_RELOC_M32C_RL_JUMP****BFD_RELOC_M32C_RL_1ADDR****BFD_RELOC_M32C_RL_2ADDR**

Renesas M16C/M32C Relocations.

BFD_RELOC_M32R_24

Renesas M32R (formerly Mitsubishi M32R) relocs. This is a 24 bit absolute address.

BFD_RELOC_M32R_10_PCREL

This is a 10-bit pc-relative reloc with the right 2 bits assumed to be 0.

BFD_RELOC_M32R_18_PCREL

This is an 18-bit reloc with the right 2 bits assumed to be 0.

BFD_RELOC_M32R_26_PCREL

This is a 26-bit reloc with the right 2 bits assumed to be 0.

BFD_RELOC_M32R_HI16_ULO

This is a 16-bit reloc containing the high 16 bits of an address used when the lower 16 bits are treated as unsigned.

BFD_RELOC_M32R_HI16_SLO

This is a 16-bit reloc containing the high 16 bits of an address used when the lower 16 bits are treated as signed.

BFD_RELOC_M32R_L016

This is a 16-bit reloc containing the lower 16 bits of an address.

BFD_RELOC_M32R_SDA16

This is a 16-bit reloc containing the small data area offset for use in add3, load, and store instructions.

BFD_RELOC_M32R_GOT24
BFD_RELOC_M32R_26_PLTREL
BFD_RELOC_M32R_COPY
BFD_RELOC_M32R_GLOB_DAT
BFD_RELOC_M32R_JMP_SLOT
BFD_RELOC_M32R_RELATIVE
BFD_RELOC_M32R_GOTOFF
BFD_RELOC_M32R_GOTOFF_HI_UL0
BFD_RELOC_M32R_GOTOFF_HI_SLO
BFD_RELOC_M32R_GOTOFF_LO
BFD_RELOC_M32R_GOTPC24
BFD_RELOC_M32R_GOT16_HI_UL0
BFD_RELOC_M32R_GOT16_HI_SLO
BFD_RELOC_M32R_GOT16_LO
BFD_RELOC_M32R_GOTPC_HI_UL0
BFD_RELOC_M32R_GOTPC_HI_SLO
BFD_RELOC_M32R_GOTPC_LO

For PIC.

BFD_RELOC_NDS32_20

NDS32 relocs. This is a 20 bit absolute address.

BFD_RELOC_NDS32_9_PCREL

This is a 9-bit pc-relative reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_WORD_9_PCREL

This is a 9-bit pc-relative reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_15_PCREL

This is an 15-bit reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_17_PCREL

This is an 17-bit reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_25_PCREL

This is a 25-bit reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_HI20

This is a 20-bit reloc containing the high 20 bits of an address used with the lower 12 bits

BFD_RELOC_NDS32_L012S3

This is a 12-bit reloc containing the lower 12 bits of an address then shift right by 3.
This is used with ldi,sdi...

BFD_RELOC_NDS32_L012S2

This is a 12-bit reloc containing the lower 12 bits of an address then shift left by 2.
This is used with lwi,swi...

BFD_RELOC_NDS32_L012S1

This is a 12-bit reloc containing the lower 12 bits of an address then shift left by 1.
This is used with lhi,shi...

BFD_RELOC_NDS32_L012S0

This is a 12-bit reloc containing the lower 12 bits of an address then shift left by 0.
This is used with lbisbi...

BFD_RELOC_NDS32_L012S0_ORI

This is a 12-bit reloc containing the lower 12 bits of an address then shift left by 0.
This is only used with branch relaxations

BFD_RELOC_NDS32_SDA15S3

This is a 15-bit reloc containing the small data area 18-bit signed offset and shift left
by 3 for use in ldi, sdi...

BFD_RELOC_NDS32_SDA15S2

This is a 15-bit reloc containing the small data area 17-bit signed offset and shift left
by 2 for use in lwi, swi...

BFD_RELOC_NDS32_SDA15S1

This is a 15-bit reloc containing the small data area 16-bit signed offset and shift left
by 1 for use in lhi, shi...

BFD_RELOC_NDS32_SDA15S0

This is a 15-bit reloc containing the small data area 15-bit signed offset and shift left
by 0 for use in lbi, sbi...

BFD_RELOC_NDS32_SDA16S3

This is a 16-bit reloc containing the small data area 16-bit signed offset and shift left
by 3

BFD_RELOC_NDS32_SDA17S2

This is a 17-bit reloc containing the small data area 17-bit signed offset and shift left
by 2 for use in lwi.gp, swi.gp...

BFD_RELOC_NDS32_SDA18S1

This is a 18-bit reloc containing the small data area 18-bit signed offset and shift left
by 1 for use in lhi.gp, shi.gp...

BFD_RELOC_NDS32_SDA19S0

This is a 19-bit reloc containing the small data area 19-bit signed offset and shift left
by 0 for use in lbi.gp, sbi.gp...

BFD_RELOC_NDS32_GOT20**BFD_RELOC_NDS32_9_PLTREL****BFD_RELOC_NDS32_25_PLTREL****BFD_RELOC_NDS32_COPY****BFD_RELOC_NDS32_GLOB_DAT****BFD_RELOC_NDS32_JMP_SLOT****BFD_RELOC_NDS32_RELATIVE**

BFD_RELOC_NDS32_GOTOFF
BFD_RELOC_NDS32_GOTOFF_HI20
BFD_RELOC_NDS32_GOTOFF_L012
BFD_RELOC_NDS32_GOTPC20
BFD_RELOC_NDS32_GOT_HI20
BFD_RELOC_NDS32_GOT_L012
BFD_RELOC_NDS32_GOTPC_HI20
BFD_RELOC_NDS32_GOTPC_L012
for PIC

BFD_RELOC_NDS32_INSN16
BFD_RELOC_NDS32_LABEL
BFD_RELOC_NDS32_LONGCALL1
BFD_RELOC_NDS32_LONGCALL2
BFD_RELOC_NDS32_LONGCALL3
BFD_RELOC_NDS32_LONGJUMP1
BFD_RELOC_NDS32_LONGJUMP2
BFD_RELOC_NDS32_LONGJUMP3
BFD_RELOC_NDS32_LOADSTORE
BFD_RELOC_NDS32_9_FIXED
BFD_RELOC_NDS32_15_FIXED
BFD_RELOC_NDS32_17_FIXED
BFD_RELOC_NDS32_25_FIXED
BFD_RELOC_NDS32_LONGCALL4
BFD_RELOC_NDS32_LONGCALL5
BFD_RELOC_NDS32_LONGCALL6
BFD_RELOC_NDS32_LONGJUMP4
BFD_RELOC_NDS32_LONGJUMP5
BFD_RELOC_NDS32_LONGJUMP6
BFD_RELOC_NDS32_LONGJUMP7
for relax

BFD_RELOC_NDS32_PLTREL_HI20
BFD_RELOC_NDS32_PLTREL_L012
BFD_RELOC_NDS32_PLT_GOTREL_HI20
BFD_RELOC_NDS32_PLT_GOTREL_L012
for PIC

BFD_RELOC_NDS32_SDA12S2_DP
BFD_RELOC_NDS32_SDA12S2_SP
BFD_RELOC_NDS32_L012S2_DP
BFD_RELOC_NDS32_L012S2_SP
for floating point

BFD_RELOC_NDS32_DWARF2_OP1
BFD_RELOC_NDS32_DWARF2_OP2
BFD_RELOC_NDS32_DWARF2_LEB
for dwarf2 debug_line.

BFD_RELOC_NDS32_UPDATE_TA

for eliminate 16-bit instructions

BFD_RELOC_NDS32_PLT_GOTREL_LO20

BFD_RELOC_NDS32_PLT_GOTREL_LO15

BFD_RELOC_NDS32_PLT_GOTREL_LO19

BFD_RELOC_NDS32_GOT_LO15

BFD_RELOC_NDS32_GOT_LO19

BFD_RELOC_NDS32_GOTOFF_LO15

BFD_RELOC_NDS32_GOTOFF_LO19

BFD_RELOC_NDS32_GOT15S2

BFD_RELOC_NDS32_GOT17S2

for PIC object relaxation

BFD_RELOC_NDS32_5

NDS32 relocs. This is a 5 bit absolute address.

BFD_RELOC_NDS32_10_UPCREL

This is a 10-bit unsigned pc-relative reloc with the right 1 bit assumed to be 0.

BFD_RELOC_NDS32_SDA_FP7U2_RELA

If fp were omitted, fp can used as another gp.

BFD_RELOC_NDS32_RELAX_ENTRY

BFD_RELOC_NDS32_GOT_SUFF

BFD_RELOC_NDS32_GOTOFF_SUFF

BFD_RELOC_NDS32_PLT_GOT_SUFF

BFD_RELOC_NDS32_MULCALL_SUFF

BFD_RELOC_NDS32_PTR

BFD_RELOC_NDS32_PTR_COUNT

BFD_RELOC_NDS32_PTR_RESOLVED

BFD_RELOC_NDS32_PLTBLOCK

BFD_RELOC_NDS32_RELAX_REGION_BEGIN

BFD_RELOC_NDS32_RELAX_REGION_END

BFD_RELOC_NDS32_MINUEND

BFD_RELOC_NDS32_SUBTRAHEND

BFD_RELOC_NDS32_DIFF8

BFD_RELOC_NDS32_DIFF16

BFD_RELOC_NDS32_DIFF32

BFD_RELOC_NDS32_DIFF_ULEB128

BFD_RELOC_NDS32_EMPTY

relaxation relative relocation types

BFD_RELOC_NDS32_25_ABS

This is a 25 bit absolute address.

BFD_RELOC_NDS32_DATA

BFD_RELOC_NDS32_TRAN

BFD_RELOC_NDS32_17IFC_PCREL

BFD_RELOC_NDS32_10IFCU_PCREL

For ex9 and ifc using.

BFD_RELOC_NDS32_TPOFF

BFD_RELOC_NDS32_GOTTPOFF

BFD_RELOC_NDS32_TLS_LE_HI20

BFD_RELOC_NDS32_TLS_LE_L012

BFD_RELOC_NDS32_TLS_LE_20

BFD_RELOC_NDS32_TLS_LE_15S0

BFD_RELOC_NDS32_TLS_LE_15S1

BFD_RELOC_NDS32_TLS_LE_15S2

BFD_RELOC_NDS32_TLS_LE_ADD

BFD_RELOC_NDS32_TLS_LE_LS

BFD_RELOC_NDS32_TLS_IE_HI20

BFD_RELOC_NDS32_TLS_IE_L012

BFD_RELOC_NDS32_TLS_IE_L012S2

BFD_RELOC_NDS32_TLS_IEGP_HI20

BFD_RELOC_NDS32_TLS_IEGP_L012

BFD_RELOC_NDS32_TLS_IEGP_L012S2

BFD_RELOC_NDS32_TLS_IEGP_LW

BFD_RELOC_NDS32_TLS_DESC

BFD_RELOC_NDS32_TLS_DESC_HI20

BFD_RELOC_NDS32_TLS_DESC_L012

BFD_RELOC_NDS32_TLS_DESC_20

BFD_RELOC_NDS32_TLS_DESC_SDA17S2

BFD_RELOC_NDS32_TLS_DESC_ADD

BFD_RELOC_NDS32_TLS_DESC_FUNC

BFD_RELOC_NDS32_TLS_DESC_CALL

BFD_RELOC_NDS32_TLS_DESC_MEM

BFD_RELOC_NDS32_REMOVE

BFD_RELOC_NDS32_GROUP

For TLS.

BFD_RELOC_NDS32_LSI

For floating load store relaxation.

BFD_RELOC_V850_9_PCREL

This is a 9-bit reloc

BFD_RELOC_V850_22_PCREL

This is a 22-bit reloc

BFD_RELOC_V850_SDA_16_16_OFFSET

This is a 16 bit offset from the short data area pointer.

BFD_RELOC_V850_SDA_15_16_OFFSET

This is a 16 bit offset (of which only 15 bits are used) from the short data area pointer.

BFD_RELOC_V850_ZDA_16_16_OFFSET

This is a 16 bit offset from the zero data area pointer.

BFD_RELOC_V850_ZDA_15_16_OFFSET

This is a 16 bit offset (of which only 15 bits are used) from the zero data area pointer.

BFD_RELOC_V850_TDA_6_8_OFFSET

This is an 8 bit offset (of which only 6 bits are used) from the tiny data area pointer.

BFD_RELOC_V850_TDA_7_8_OFFSET

This is an 8bit offset (of which only 7 bits are used) from the tiny data area pointer.

BFD_RELOC_V850_TDA_7_7_OFFSET

This is a 7 bit offset from the tiny data area pointer.

BFD_RELOC_V850_TDA_16_16_OFFSET

This is a 16 bit offset from the tiny data area pointer.

BFD_RELOC_V850_TDA_4_5_OFFSET

This is a 5 bit offset (of which only 4 bits are used) from the tiny data area pointer.

BFD_RELOC_V850_TDA_4_4_OFFSET

This is a 4 bit offset from the tiny data area pointer.

BFD_RELOC_V850_SDA_16_16_SPLIT_OFFSET

This is a 16 bit offset from the short data area pointer, with the bits placed non-contiguously in the instruction.

BFD_RELOC_V850_ZDA_16_16_SPLIT_OFFSET

This is a 16 bit offset from the zero data area pointer, with the bits placed non-contiguously in the instruction.

BFD_RELOC_V850_CALLT_6_7_OFFSET

This is a 6 bit offset from the call table base pointer.

BFD_RELOC_V850_CALLT_16_16_OFFSET

This is a 16 bit offset from the call table base pointer.

BFD_RELOC_V850_LONGCALL

Used for relaxing indirect function calls.

BFD_RELOC_V850_LONGJUMP

Used for relaxing indirect jumps.

BFD_RELOC_V850_ALIGN

Used to maintain alignment whilst relaxing.

BFD_RELOC_V850_LO16_SPLIT_OFFSET

This is a variation of BFD_RELOC_LO16 that can be used in v850e ld.bu instructions.

BFD_RELOC_V850_16_PCREL

This is a 16-bit reloc.

BFD_RELOC_V850_17_PCREL

This is a 17-bit reloc.

BFD_RELOC_V850_23

This is a 23-bit reloc.

BFD_RELOC_V850_32_PCREL

This is a 32-bit reloc.

BFD_RELOC_V850_32_ABS

This is a 32-bit reloc.

BFD_RELOC_V850_16_SPLIT_OFFSET

This is a 16-bit reloc.

BFD_RELOC_V850_16_S1

This is a 16-bit reloc.

BFD_RELOC_V850_L016_S1

Low 16 bits. 16 bit shifted by 1.

BFD_RELOC_V850_CALLT_15_16_OFFSET

This is a 16 bit offset from the call table base pointer.

BFD_RELOC_V850_32_GOTPCREL

DSO relocations.

BFD_RELOC_V850_16_GOT

DSO relocations.

BFD_RELOC_V850_32_GOT

DSO relocations.

BFD_RELOC_V850_22_PLT_PCREL

DSO relocations.

BFD_RELOC_V850_32_PLT_PCREL

DSO relocations.

BFD_RELOC_V850_COPY

DSO relocations.

BFD_RELOC_V850_GLOB_DAT

DSO relocations.

BFD_RELOC_V850_JMP_SLOT

DSO relocations.

BFD_RELOC_V850_RELATIVE

DSO relocations.

BFD_RELOC_V850_16_GOTOFF

DSO relocations.

BFD_RELOC_V850_32_GOTOFF

DSO relocations.

BFD_RELOC_V850_CODE

start code.

BFD_RELOC_V850_DATA

start data in text.

BFD_RELOC_TIC30_LDP

This is a 8bit DP reloc for the tms320c30, where the most significant 8 bits of a 24 bit word are placed into the least significant 8 bits of the opcode.

BFD_RELOC_TIC54X_PARTLS7

This is a 7bit reloc for the tms320c54x, where the least significant 7 bits of a 16 bit word are placed into the least significant 7 bits of the opcode.

BFD_RELOC_TIC54X_PARTMS9

This is a 9bit DP reloc for the tms320c54x, where the most significant 9 bits of a 16 bit word are placed into the least significant 9 bits of the opcode.

BFD_RELOC_TIC54X_23

This is an extended address 23-bit reloc for the tms320c54x.

BFD_RELOC_TIC54X_16_OF_23

This is a 16-bit reloc for the tms320c54x, where the least significant 16 bits of a 23-bit extended address are placed into the opcode.

BFD_RELOC_TIC54X_MS7_OF_23

This is a reloc for the tms320c54x, where the most significant 7 bits of a 23-bit extended address are placed into the opcode.

BFD_RELOC_C6000_PCR_S21

BFD_RELOC_C6000_PCR_S12

BFD_RELOC_C6000_PCR_S10

BFD_RELOC_C6000_PCR_S7

BFD_RELOC_C6000_ABS_S16

BFD_RELOC_C6000_ABS_L16

BFD_RELOC_C6000_ABS_H16

BFD_RELOC_C6000_SBR_U15_B

BFD_RELOC_C6000_SBR_U15_H

BFD_RELOC_C6000_SBR_U15_W

BFD_RELOC_C6000_SBR_S16

BFD_RELOC_C6000_SBR_L16_B

BFD_RELOC_C6000_SBR_L16_H

BFD_RELOC_C6000_SBR_L16_W

BFD_RELOC_C6000_SBR_H16_B

BFD_RELOC_C6000_SBR_H16_H

BFD_RELOC_C6000_SBR_H16_W

BFD_RELOC_C6000_SBR_GOT_U15_W

BFD_RELOC_C6000_SBR_GOT_L16_W

BFD_RELOC_C6000_SBR_GOT_H16_W

BFD_RELOC_C6000_DSBT_INDEX

BFD_RELOC_C6000_PREL31
BFD_RELOC_C6000_COPY
BFD_RELOC_C6000_JUMP_SLOT
BFD_RELOC_C6000_EHTYPE
BFD_RELOC_C6000_PCR_H16
BFD_RELOC_C6000_PCR_L16
BFD_RELOC_C6000_ALIGN
BFD_RELOC_C6000_FPHEAD
BFD_RELOC_C6000_NOCMP

TMS320C6000 relocations.

BFD_RELOC_FR30_48

This is a 48 bit reloc for the FR30 that stores 32 bits.

BFD_RELOC_FR30_20

This is a 32 bit reloc for the FR30 that stores 20 bits split up into two sections.

BFD_RELOC_FR30_6_IN_4

This is a 16 bit reloc for the FR30 that stores a 6 bit word offset in 4 bits.

BFD_RELOC_FR30_8_IN_8

This is a 16 bit reloc for the FR30 that stores an 8 bit byte offset into 8 bits.

BFD_RELOC_FR30_9_IN_8

This is a 16 bit reloc for the FR30 that stores a 9 bit short offset into 8 bits.

BFD_RELOC_FR30_10_IN_8

This is a 16 bit reloc for the FR30 that stores a 10 bit word offset into 8 bits.

BFD_RELOC_FR30_9_PCREL

This is a 16 bit reloc for the FR30 that stores a 9 bit pc relative short offset into 8 bits.

BFD_RELOC_FR30_12_PCREL

This is a 16 bit reloc for the FR30 that stores a 12 bit pc relative short offset into 11 bits.

BFD_RELOC_MCORE_PCREL_IMM8BY4

BFD_RELOC_MCORE_PCREL_IMM11BY2

BFD_RELOC_MCORE_PCREL_IMM4BY2

BFD_RELOC_MCORE_PCREL_32

BFD_RELOC_MCORE_PCREL_JSR_IMM11BY2

BFD_RELOC_MCORE_RVA

Motorola Mcore relocations.

BFD_RELOC_MEP_8

BFD_RELOC_MEP_16

BFD_RELOC_MEP_32

BFD_RELOC_MEP_PCREL8A2

BFD_RELOC_MEP_PCREL12A2

BFD_RELOC_MEP_PCREL17A2
BFD_RELOC_MEP_PCREL24A2
BFD_RELOC_MEP_PCABS24A2
BFD_RELOC_MEP_LOW16
BFD_RELOC_MEP_HI16U
BFD_RELOC_MEP_HI16S
BFD_RELOC_MEP_GPREL
BFD_RELOC_MEP_TPREL
BFD_RELOC_MEP_TPREL7
BFD_RELOC_MEP_TPREL7A2
BFD_RELOC_MEP_TPREL7A4
BFD_RELOC_MEP_UIMM24
BFD_RELOC_MEP_ADDR24A4
BFD_RELOC_MEP_GNU_VTINHERIT
BFD_RELOC_MEP_GNU_VTENTRY

Toshiba Media Processor Relocations.

BFD_RELOC_METAG_HIADDR16
BFD_RELOC_METAG_LOADADDR16
BFD_RELOC_METAG_RELBRANCH
BFD_RELOC_METAG_GETSETOFF
BFD_RELOC_METAG_HIOG
BFD_RELOC_METAG_LOOG
BFD_RELOC_METAG_REL8
BFD_RELOC_METAG_REL16
BFD_RELOC_METAG_HI16_GOTOFF
BFD_RELOC_METAG_LO16_GOTOFF
BFD_RELOC_METAG_GETSET_GOTOFF
BFD_RELOC_METAG_GETSET_GOT
BFD_RELOC_METAG_HI16_GOTPC
BFD_RELOC_METAG_LO16_GOTPC
BFD_RELOC_METAG_HI16_PLT
BFD_RELOC_METAG_LO16_PLT
BFD_RELOC_METAG_RELBRANCH_PLT
BFD_RELOC_METAG_GOTOFF
BFD_RELOC_METAG_PLT
BFD_RELOC_METAG_COPY
BFD_RELOC_METAG_JMP_SLOT
BFD_RELOC_METAG_RELATIVE
BFD_RELOC_METAG_GLOB_DAT
BFD_RELOC_METAG_TLS_GD
BFD_RELOC_METAG_TLS_LDM
BFD_RELOC_METAG_TLS_LDO_HI16
BFD_RELOC_METAG_TLS_LDO_LO16
BFD_RELOC_METAG_TLS_LDO
BFD_RELOC_METAG_TLS_IE
BFD_RELOC_METAG_TLS_IENONPIC

BFD_RELOC_METAG_TLS_IENONPIC_HI16
BFD_RELOC_METAG_TLS_IENONPIC_LO16
BFD_RELOC_METAG_TLS_TPOFF
BFD_RELOC_METAG_TLS_DTPMOD
BFD_RELOC_METAG_TLS_DTPOFF
BFD_RELOC_METAG_TLS_LE
BFD_RELOC_METAG_TLS_LE_HI16
BFD_RELOC_METAG_TLS_LE_LO16

Imagination Technologies Meta relocations.

BFD_RELOC_MMIX_GETA
BFD_RELOC_MMIX_GETA_1
BFD_RELOC_MMIX_GETA_2
BFD_RELOC_MMIX_GETA_3

These are relocations for the GETA instruction.

BFD_RELOC_MMIX_CBRANCH
BFD_RELOC_MMIX_CBRANCH_J
BFD_RELOC_MMIX_CBRANCH_1
BFD_RELOC_MMIX_CBRANCH_2
BFD_RELOC_MMIX_CBRANCH_3

These are relocations for a conditional branch instruction.

BFD_RELOC_MMIX_PUSHJ
BFD_RELOC_MMIX_PUSHJ_1
BFD_RELOC_MMIX_PUSHJ_2
BFD_RELOC_MMIX_PUSHJ_3
BFD_RELOC_MMIX_PUSHJ_STUBBABLE

These are relocations for the PUSHJ instruction.

BFD_RELOC_MMIX_JMP
BFD_RELOC_MMIX_JMP_1
BFD_RELOC_MMIX_JMP_2
BFD_RELOC_MMIX_JMP_3

These are relocations for the JMP instruction.

BFD_RELOC_MMIX_ADDR19

This is a relocation for a relative address as in a GETA instruction or a branch.

BFD_RELOC_MMIX_ADDR27

This is a relocation for a relative address as in a JMP instruction.

BFD_RELOC_MMIX_REG_OR_BYTE

This is a relocation for an instruction field that may be a general register or a value 0..255.

BFD_RELOC_MMIX_REG

This is a relocation for an instruction field that may be a general register.

BFD_RELOC_MMIX_BASE_PLUS_OFFSET

This is a relocation for two instruction fields holding a register and an offset, the equivalent of the relocation.

BFD_RELOC_MMIX_LOCAL

This relocation is an assertion that the expression is not allocated as a global register. It does not modify contents.

BFD_RELOC_AVR_7_PCREL

This is a 16 bit reloc for the AVR that stores 8 bit pc relative short offset into 7 bits.

BFD_RELOC_AVR_13_PCREL

This is a 16 bit reloc for the AVR that stores 13 bit pc relative short offset into 12 bits.

BFD_RELOC_AVR_16_PM

This is a 16 bit reloc for the AVR that stores 17 bit value (usually program memory address) into 16 bits.

BFD_RELOC_AVR_LO8_LDI

This is a 16 bit reloc for the AVR that stores 8 bit value (usually data memory address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_HI8_LDI

This is a 16 bit reloc for the AVR that stores 8 bit value (high 8 bit of data memory address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_HH8_LDI

This is a 16 bit reloc for the AVR that stores 8 bit value (most high 8 bit of program memory address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_MS8_LDI

This is a 16 bit reloc for the AVR that stores 8 bit value (most high 8 bit of 32 bit value) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_LO8_LDI_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (usually data memory address) into 8 bit immediate value of SUBI insn.

BFD_RELOC_AVR_HI8_LDI_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (high 8 bit of data memory address) into 8 bit immediate value of SUBI insn.

BFD_RELOC_AVR_HH8_LDI_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (most high 8 bit of program memory address) into 8 bit immediate value of LDI or SUBI insn.

BFD_RELOC_AVR_MS8_LDI_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (msb of 32 bit value) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_LO8_LDI_PM

This is a 16 bit reloc for the AVR that stores 8 bit value (usually command address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_LO8_LDI_GS

This is a 16 bit reloc for the AVR that stores 8 bit value (command address) into 8 bit immediate value of LDI insn. If the address is beyond the 128k boundary, the linker inserts a jump stub for this reloc in the lower 128k.

BFD_RELOC_AVR_HI8_LDI_PM

This is a 16 bit reloc for the AVR that stores 8 bit value (high 8 bit of command address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_HI8_LDI_GS

This is a 16 bit reloc for the AVR that stores 8 bit value (high 8 bit of command address) into 8 bit immediate value of LDI insn. If the address is beyond the 128k boundary, the linker inserts a jump stub for this reloc below 128k.

BFD_RELOC_AVR_HH8_LDI_PM

This is a 16 bit reloc for the AVR that stores 8 bit value (most high 8 bit of command address) into 8 bit immediate value of LDI insn.

BFD_RELOC_AVR_LO8_LDI_PM_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (usually command address) into 8 bit immediate value of SUBI insn.

BFD_RELOC_AVR_HI8_LDI_PM_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (high 8 bit of 16 bit command address) into 8 bit immediate value of SUBI insn.

BFD_RELOC_AVR_HH8_LDI_PM_NEG

This is a 16 bit reloc for the AVR that stores negated 8 bit value (high 6 bit of 22 bit command address) into 8 bit immediate value of SUBI insn.

BFD_RELOC_AVR_CALL

This is a 32 bit reloc for the AVR that stores 23 bit value into 22 bits.

BFD_RELOC_AVR_LDI

This is a 16 bit reloc for the AVR that stores all needed bits for absolute addressing with ldi with overflow check to linktime

BFD_RELOC_AVR_6

This is a 6 bit reloc for the AVR that stores offset for ldd/std instructions

BFD_RELOC_AVR_6_ADIW

This is a 6 bit reloc for the AVR that stores offset for adiw/sbiw instructions

BFD_RELOC_AVR_8_LO

This is a 8 bit reloc for the AVR that stores bits 0..7 of a symbol in .byte lo8(symbol)

BFD_RELOC_AVR_8_HI

This is a 8 bit reloc for the AVR that stores bits 8..15 of a symbol in .byte hi8(symbol)

BFD_RELOC_AVR_8_HLO

This is a 8 bit reloc for the AVR that stores bits 16..23 of a symbol in .byte hlo8(symbol)

BFD_RELOC_AVR_DIFF8**BFD_RELOC_AVR_DIFF16****BFD_RELOC_AVR_DIFF32**

AVR relocations to mark the difference of two local symbols. These are only needed to support linker relaxation and can be ignored when not relaxing. The field is set to the value of the difference assuming no relaxation. The relocation encodes the position of the second symbol so the linker can determine whether to adjust the field value.

BFD_RELOC_AVR_LDS_STS_16

This is a 7 bit reloc for the AVR that stores SRAM address for 16bit lds and sts instructions supported only tiny core.

BFD_RELOC_AVR_PORT6

This is a 6 bit reloc for the AVR that stores an I/O register number for the IN and OUT instructions

BFD_RELOC_AVR_PORT5

This is a 5 bit reloc for the AVR that stores an I/O register number for the SBIC, SBIS, SBI and CBI instructions

BFD_RELOC_RISCV_HI20**BFD_RELOC_RISCV_PCREL_HI20****BFD_RELOC_RISCV_PCREL_L012_I****BFD_RELOC_RISCV_PCREL_L012_S****BFD_RELOC_RISCV_L012_I****BFD_RELOC_RISCV_L012_S****BFD_RELOC_RISCV_GPREL12_I****BFD_RELOC_RISCV_GPREL12_S****BFD_RELOC_RISCV_TPREL_HI20****BFD_RELOC_RISCV_TPREL_L012_I****BFD_RELOC_RISCV_TPREL_L012_S****BFD_RELOC_RISCV_TPREL_ADD****BFD_RELOC_RISCV_CALL****BFD_RELOC_RISCV_CALL_PLT****BFD_RELOC_RISCV_ADD8****BFD_RELOC_RISCV_ADD16****BFD_RELOC_RISCV_ADD32****BFD_RELOC_RISCV_ADD64****BFD_RELOC_RISCV_SUB8****BFD_RELOC_RISCV_SUB16****BFD_RELOC_RISCV_SUB32****BFD_RELOC_RISCV_SUB64****BFD_RELOC_RISCV_GOT_HI20****BFD_RELOC_RISCV_TLS_GOT_HI20****BFD_RELOC_RISCV_TLS_GD_HI20****BFD_RELOC_RISCV_JMP**

BFD_RELOC_RISCV_TLS_DTPMOD32
BFD_RELOC_RISCV_TLS_DTPREL32
BFD_RELOC_RISCV_TLS_DTPMOD64
BFD_RELOC_RISCV_TLS_DTPREL64
BFD_RELOC_RISCV_TLS_TPREL32
BFD_RELOC_RISCV_TLS_TPREL64
BFD_RELOC_RISCV_ALIGN
BFD_RELOC_RISCV_RVC_BRANCH
BFD_RELOC_RISCV_RVC_JUMP
BFD_RELOC_RISCV_RVC_LUI
BFD_RELOC_RISCV_GPREL_I
BFD_RELOC_RISCV_GPREL_S
BFD_RELOC_RISCV_TPREL_I
BFD_RELOC_RISCV_TPREL_S
BFD_RELOC_RISCV_RELAX
BFD_RELOC_RISCV_CFA
BFD_RELOC_RISCV_SUB6
BFD_RELOC_RISCV_SET6
BFD_RELOC_RISCV_SET8
BFD_RELOC_RISCV_SET16
BFD_RELOC_RISCV_SET32
BFD_RELOC_RISCV_32_PCREL
BFD_RELOC_RISCV_SET_ULEB128
BFD_RELOC_RISCV_SUB_ULEB128
RISC-V relocations.

BFD_RELOC_RL78_NEG8
BFD_RELOC_RL78_NEG16
BFD_RELOC_RL78_NEG24
BFD_RELOC_RL78_NEG32
BFD_RELOC_RL78_16_OP
BFD_RELOC_RL78_24_OP
BFD_RELOC_RL78_32_OP
BFD_RELOC_RL78_8U
BFD_RELOC_RL78_16U
BFD_RELOC_RL78_24U
BFD_RELOC_RL78_DIR3U_PCREL
BFD_RELOC_RL78_DIFF
BFD_RELOC_RL78_GPRELB
BFD_RELOC_RL78_GPRELW
BFD_RELOC_RL78_GPRELL
BFD_RELOC_RL78_SYM
BFD_RELOC_RL78_OP_SUBTRACT
BFD_RELOC_RL78_OP_NEG
BFD_RELOC_RL78_OP_AND
BFD_RELOC_RL78_OP_SHRA
BFD_RELOC_RL78_ABS8

BFD_RELOC_RL78_ABS16
BFD_RELOC_RL78_ABS16_REV
BFD_RELOC_RL78_ABS32
BFD_RELOC_RL78_ABS32_REV
BFD_RELOC_RL78_ABS16U
BFD_RELOC_RL78_ABS16UW
BFD_RELOC_RL78_ABS16UL
BFD_RELOC_RL78_RELAX
BFD_RELOC_RL78_HI16
BFD_RELOC_RL78_HI8
BFD_RELOC_RL78_L016
BFD_RELOC_RL78_CODE
BFD_RELOC_RL78_SADDR
Renesas RL78 Relocations.

BFD_RELOC_RX_NEG8
BFD_RELOC_RX_NEG16
BFD_RELOC_RX_NEG24
BFD_RELOC_RX_NEG32
BFD_RELOC_RX_16_OP
BFD_RELOC_RX_24_OP
BFD_RELOC_RX_32_OP
BFD_RELOC_RX_8U
BFD_RELOC_RX_16U
BFD_RELOC_RX_24U
BFD_RELOC_RX_DIR3U_PCREL
BFD_RELOC_RX_DIFF
BFD_RELOC_RX_GPRELB
BFD_RELOC_RX_GPRELW
BFD_RELOC_RX_GPRELL
BFD_RELOC_RX_SYM
BFD_RELOC_RX_OP_SUBTRACT
BFD_RELOC_RX_OP_NEG
BFD_RELOC_RX_ABS8
BFD_RELOC_RX_ABS16
BFD_RELOC_RX_ABS16_REV
BFD_RELOC_RX_ABS32
BFD_RELOC_RX_ABS32_REV
BFD_RELOC_RX_ABS16U
BFD_RELOC_RX_ABS16UW
BFD_RELOC_RX_ABS16UL
BFD_RELOC_RX_RELAX
Renesas RX Relocations.

BFD_RELOC_390_12
Direct 12 bit.

BFD_RELOC_390_GOT12
12 bit GOT offset.

BFD_RELOC_390_PLT32
32 bit PC relative PLT address.

BFD_RELOC_390_COPY
Copy symbol at runtime.

BFD_RELOC_390_GLOB_DAT
Create GOT entry.

BFD_RELOC_390_JMP_SLOT
Create PLT entry.

BFD_RELOC_390_RELATIVE
Adjust by program base.

BFD_RELOC_390_GOTPC
32 bit PC relative offset to GOT.

BFD_RELOC_390_GOT16
16 bit GOT offset.

BFD_RELOC_390_PC12DBL
PC relative 12 bit shifted by 1.

BFD_RELOC_390_PLT12DBL
12 bit PC rel. PLT shifted by 1.

BFD_RELOC_390_PC16DBL
PC relative 16 bit shifted by 1.

BFD_RELOC_390_PLT16DBL
16 bit PC rel. PLT shifted by 1.

BFD_RELOC_390_PC24DBL
PC relative 24 bit shifted by 1.

BFD_RELOC_390_PLT24DBL
24 bit PC rel. PLT shifted by 1.

BFD_RELOC_390_PC32DBL
PC relative 32 bit shifted by 1.

BFD_RELOC_390_PLT32DBL
32 bit PC rel. PLT shifted by 1.

BFD_RELOC_390_GOTPCDBL
32 bit PC rel. GOT shifted by 1.

BFD_RELOC_390_GOT64
64 bit GOT offset.

BFD_RELOC_390_PLT64
64 bit PC relative PLT address.

BFD_RELOC_390_GOTENT
32 bit rel. offset to GOT entry.

BFD_RELOC_390_GOTOFF64
64 bit offset to GOT.

BFD_RELOC_390_GOTPLT12
12-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_390_GOTPLT16
16-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_390_GOTPLT32
32-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_390_GOTPLT64
64-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_390_GOTPLTENT
32-bit rel. offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_390_PLTOFF16
16-bit rel. offset from the GOT to a PLT entry.

BFD_RELOC_390_PLTOFF32
32-bit rel. offset from the GOT to a PLT entry.

BFD_RELOC_390_PLTOFF64
64-bit rel. offset from the GOT to a PLT entry.

BFD_RELOC_390_TLS_LOAD

BFD_RELOC_390_TLS_GDCALL

BFD_RELOC_390_TLS_LDCALL

BFD_RELOC_390_TLS_GD32

BFD_RELOC_390_TLS_GD64

BFD_RELOC_390_TLS_GOTIE12

BFD_RELOC_390_TLS_GOTIE32

BFD_RELOC_390_TLS_GOTIE64

BFD_RELOC_390_TLS_LDM32

BFD_RELOC_390_TLS_LDM64

BFD_RELOC_390_TLS_IE32

BFD_RELOC_390_TLS_IE64

BFD_RELOC_390_TLS_IEENT

BFD_RELOC_390_TLS_LE32

BFD_RELOC_390_TLS_LE64

BFD_RELOC_390_TLS_LD032

BFD_RELOC_390_TLS_LD064

BFD_RELOC_390_TLS_DTPMOD

BFD_RELOC_390_TLS_DTPOFF

BFD_RELOC_390_TLS_TPOFF

s390 tls relocations.

BFD_RELOC_390_20

BFD_RELOC_390_GOT20

BFD_RELOC_390_GOTPLT20

BFD_RELOC_390_TLS_GOTIE20

Long displacement extension.

BFD_RELOC_390_IRELATIVE

STT_GNU_IFUNC relocation.

BFD_RELOC_SCORE_GPREL15

Score relocations Low 16 bit for load/store

BFD_RELOC_SCORE_DUMMY2

BFD_RELOC_SCORE_JMP

This is a 24-bit reloc with the right 1 bit assumed to be 0

BFD_RELOC_SCORE_BRANCH

This is a 19-bit reloc with the right 1 bit assumed to be 0

BFD_RELOC_SCORE_IMM30

This is a 32-bit reloc for 48-bit instructions.

BFD_RELOC_SCORE_IMM32

This is a 32-bit reloc for 48-bit instructions.

BFD_RELOC_SCORE16_JMP

This is a 11-bit reloc with the right 1 bit assumed to be 0

BFD_RELOC_SCORE16_BRANCH

This is a 8-bit reloc with the right 1 bit assumed to be 0

BFD_RELOC_SCORE_BCMP

This is a 9-bit reloc with the right 1 bit assumed to be 0

BFD_RELOC_SCORE_GOT15

BFD_RELOC_SCORE_GOT_L016

BFD_RELOC_SCORE_CALL15

BFD_RELOC_SCORE_DUMMY_HI16

Undocumented Score relocs

BFD_RELOC_IP2K_FR9

Scenix IP2K - 9-bit register number / data address

BFD_RELOC_IP2K_BANK

Scenix IP2K - 4-bit register/data bank number

BFD_RELOC_IP2K_ADDR16CJP

Scenix IP2K - low 13 bits of instruction word address

BFD_RELOC_IP2K_PAGE3

Scenix IP2K - high 3 bits of instruction word address

BFD_RELOC_IP2K_LO8DATA

BFD_RELOC_IP2K_HI8DATA

BFD_RELOC_IP2K_EX8DATA

Scenix IP2K - ext/low/high 8 bits of data address

BFD_RELOC_IP2K_LO8INSN

BFD_RELOC_IP2K_HI8INSN

Scenix IP2K - low/high 8 bits of instruction word address

BFD_RELOC_IP2K_PC_SKIP

Scenix IP2K - even/odd PC modifier to modify snb pcl.0

BFD_RELOC_IP2K_TEXT

Scenix IP2K - 16 bit word address in text section.

BFD_RELOC_IP2K_FR_OFFSET

Scenix IP2K - 7-bit sp or dp offset

BFD_RELOC_VPE4KMATH_DATA

BFD_RELOC_VPE4KMATH_INSN

Scenix VPE4K coprocessor - data/insn-space addressing

BFD_RELOC_VTABLE_INHERIT

BFD_RELOC_VTABLE_ENTRY

These two relocations are used by the linker to determine which of the entries in a C++ virtual function table are actually used. When the `-gc-sections` option is given, the linker will zero out the entries that are not used, so that the code for those functions need not be included in the output.

`VTABLE_INHERIT` is a zero-space relocation used to describe to the linker the inheritance tree of a C++ virtual function table. The relocation's symbol should be the parent class' vtable, and the relocation should be located at the child vtable.

`VTABLE_ENTRY` is a zero-space relocation that describes the use of a virtual function table entry. The reloc's symbol should refer to the table of the class mentioned in the code. Off of that base, an offset describes the entry that is being used. For Rel_a hosts, this offset is stored in the reloc's addend. For Rel_e hosts, we are forced to put this offset in the reloc's section offset.

BFD_RELOC_IA64_IMM14

BFD_RELOC_IA64_IMM22

BFD_RELOC_IA64_IMM64

BFD_RELOC_IA64_DIR32MSB

BFD_RELOC_IA64_DIR32LSB

BFD_RELOC_IA64_DIR64MSB

BFD_RELOC_IA64_DIR64LSB

BFD_RELOC_IA64_GPREL22

BFD_RELOC_IA64_GPREL64I

BFD_RELOC_IA64_GPREL32MSB
BFD_RELOC_IA64_GPREL32LSB
BFD_RELOC_IA64_GPREL64MSB
BFD_RELOC_IA64_GPREL64LSB
BFD_RELOC_IA64_LTOFF22
BFD_RELOC_IA64_LTOFF64I
BFD_RELOC_IA64_PLTOFF22
BFD_RELOC_IA64_PLTOFF64I
BFD_RELOC_IA64_PLTOFF64MSB
BFD_RELOC_IA64_PLTOFF64LSB
BFD_RELOC_IA64_FPTR64I
BFD_RELOC_IA64_FPTR32MSB
BFD_RELOC_IA64_FPTR32LSB
BFD_RELOC_IA64_FPTR64MSB
BFD_RELOC_IA64_FPTR64LSB
BFD_RELOC_IA64_PCREL21B
BFD_RELOC_IA64_PCREL21BI
BFD_RELOC_IA64_PCREL21M
BFD_RELOC_IA64_PCREL21F
BFD_RELOC_IA64_PCREL22
BFD_RELOC_IA64_PCREL60B
BFD_RELOC_IA64_PCREL64I
BFD_RELOC_IA64_PCREL32MSB
BFD_RELOC_IA64_PCREL32LSB
BFD_RELOC_IA64_PCREL64MSB
BFD_RELOC_IA64_PCREL64LSB
BFD_RELOC_IA64_LTOFF_FPTR22
BFD_RELOC_IA64_LTOFF_FPTR64I
BFD_RELOC_IA64_LTOFF_FPTR32MSB
BFD_RELOC_IA64_LTOFF_FPTR32LSB
BFD_RELOC_IA64_LTOFF_FPTR64MSB
BFD_RELOC_IA64_LTOFF_FPTR64LSB
BFD_RELOC_IA64_SEGREL32MSB
BFD_RELOC_IA64_SEGREL32LSB
BFD_RELOC_IA64_SEGREL64MSB
BFD_RELOC_IA64_SEGREL64LSB
BFD_RELOC_IA64_SECREL32MSB
BFD_RELOC_IA64_SECREL32LSB
BFD_RELOC_IA64_SECREL64MSB
BFD_RELOC_IA64_SECREL64LSB
BFD_RELOC_IA64_REL32MSB
BFD_RELOC_IA64_REL32LSB
BFD_RELOC_IA64_REL64MSB
BFD_RELOC_IA64_REL64LSB
BFD_RELOC_IA64_LTV32MSB
BFD_RELOC_IA64_LTV32LSB
BFD_RELOC_IA64_LTV64MSB

BFD_RELOC_IA64_LTV64LSB
BFD_RELOC_IA64_IPLTMSB
BFD_RELOC_IA64_IPLTLSB
BFD_RELOC_IA64_COPY
BFD_RELOC_IA64_LTOFF22X
BFD_RELOC_IA64_LDXMOV
BFD_RELOC_IA64_TPREL14
BFD_RELOC_IA64_TPREL22
BFD_RELOC_IA64_TPREL64I
BFD_RELOC_IA64_TPREL64MSB
BFD_RELOC_IA64_TPREL64LSB
BFD_RELOC_IA64_LTOFF_TPREL22
BFD_RELOC_IA64_DTPMOD64MSB
BFD_RELOC_IA64_DTPMOD64LSB
BFD_RELOC_IA64_LTOFF_DTPMOD22
BFD_RELOC_IA64_DTPREL14
BFD_RELOC_IA64_DTPREL22
BFD_RELOC_IA64_DTPREL64I
BFD_RELOC_IA64_DTPREL32MSB
BFD_RELOC_IA64_DTPREL32LSB
BFD_RELOC_IA64_DTPREL64MSB
BFD_RELOC_IA64_DTPREL64LSB
BFD_RELOC_IA64_LTOFF_DTPREL22

Intel IA64 Relocations.

BFD_RELOC_M68HC11_HI8

Motorola 68HC11 reloc. This is the 8 bit high part of an absolute address.

BFD_RELOC_M68HC11_LO8

Motorola 68HC11 reloc. This is the 8 bit low part of an absolute address.

BFD_RELOC_M68HC11_3B

Motorola 68HC11 reloc. This is the 3 bit of a value.

BFD_RELOC_M68HC11_RL_JUMP

Motorola 68HC11 reloc. This reloc marks the beginning of a jump/call instruction. It is used for linker relaxation to correctly identify beginning of instruction and change some branches to use PC-relative addressing mode.

BFD_RELOC_M68HC11_RL_GROUP

Motorola 68HC11 reloc. This reloc marks a group of several instructions that gcc generates and for which the linker relaxation pass can modify and/or remove some of them.

BFD_RELOC_M68HC11_LO16

Motorola 68HC11 reloc. This is the 16-bit lower part of an address. It is used for 'call' instruction to specify the symbol address without any special transformation (due to memory bank window).

BFD_RELOC_M68HC11_PAGE

Motorola 68HC11 reloc. This is a 8-bit reloc that specifies the page number of an address. It is used by 'call' instruction to specify the page number of the symbol.

BFD_RELOC_M68HC11_24

Motorola 68HC11 reloc. This is a 24-bit reloc that represents the address with a 16-bit value and a 8-bit page number. The symbol address is transformed to follow the 16K memory bank of 68HC12 (seen as mapped in the window).

BFD_RELOC_M68HC12_5B

Motorola 68HC12 reloc. This is the 5 bits of a value.

BFD_RELOC_XGATE_RL_JUMP

Freescale XGATE reloc. This reloc marks the beginning of a bra/jal instruction.

BFD_RELOC_XGATE_RL_GROUP

Freescale XGATE reloc. This reloc marks a group of several instructions that gcc generates and for which the linker relaxation pass can modify and/or remove some of them.

BFD_RELOC_XGATE_L016

Freescale XGATE reloc. This is the 16-bit lower part of an address. It is used for the '16-bit' instructions.

BFD_RELOC_XGATE_GPAGE

Freescale XGATE reloc.

BFD_RELOC_XGATE_24

Freescale XGATE reloc.

BFD_RELOC_XGATE_PCREL_9

Freescale XGATE reloc. This is a 9-bit pc-relative reloc.

BFD_RELOC_XGATE_PCREL_10

Freescale XGATE reloc. This is a 10-bit pc-relative reloc.

BFD_RELOC_XGATE_IMM8_LO

Freescale XGATE reloc. This is the 16-bit lower part of an address. It is used for the '16-bit' instructions.

BFD_RELOC_XGATE_IMM8_HI

Freescale XGATE reloc. This is the 16-bit higher part of an address. It is used for the '16-bit' instructions.

BFD_RELOC_XGATE_IMM3

Freescale XGATE reloc. This is a 3-bit pc-relative reloc.

BFD_RELOC_XGATE_IMM4

Freescale XGATE reloc. This is a 4-bit pc-relative reloc.

BFD_RELOC_XGATE_IMM5

Freescale XGATE reloc. This is a 5-bit pc-relative reloc.

BFD_RELOC_M68HC12_9B

Motorola 68HC12 reloc. This is the 9 bits of a value.

BFD_RELOC_M68HC12_16B

Motorola 68HC12 reloc. This is the 16 bits of a value.

BFD_RELOC_M68HC12_9_PCREL

Motorola 68HC12/XGATE reloc. This is a PCREL9 branch.

BFD_RELOC_M68HC12_10_PCREL

Motorola 68HC12/XGATE reloc. This is a PCREL10 branch.

BFD_RELOC_M68HC12_LO8XG

Motorola 68HC12/XGATE reloc. This is the 8 bit low part of an absolute address and immediately precedes a matching HI8XG part.

BFD_RELOC_M68HC12_HI8XG

Motorola 68HC12/XGATE reloc. This is the 8 bit high part of an absolute address and immediately follows a matching LO8XG part.

BFD_RELOC_S12Z_15_PCREL

Freescale S12Z reloc. This is a 15 bit relative address. If the most significant bits are all zero then it may be truncated to 8 bits.

BFD_RELOC_CR16_NUM8**BFD_RELOC_CR16_NUM16****BFD_RELOC_CR16_NUM32****BFD_RELOC_CR16_NUM32a****BFD_RELOC_CR16_REGRELO****BFD_RELOC_CR16_REGREL4****BFD_RELOC_CR16_REGREL4a****BFD_RELOC_CR16_REGREL14****BFD_RELOC_CR16_REGREL14a****BFD_RELOC_CR16_REGREL16****BFD_RELOC_CR16_REGREL20****BFD_RELOC_CR16_REGREL20a****BFD_RELOC_CR16_ABS20****BFD_RELOC_CR16_ABS24****BFD_RELOC_CR16_IMM4****BFD_RELOC_CR16_IMM8****BFD_RELOC_CR16_IMM16****BFD_RELOC_CR16_IMM20****BFD_RELOC_CR16_IMM24****BFD_RELOC_CR16_IMM32****BFD_RELOC_CR16_IMM32a****BFD_RELOC_CR16_DISP4****BFD_RELOC_CR16_DISP8****BFD_RELOC_CR16_DISP16****BFD_RELOC_CR16_DISP20**

BFD_RELOC_CR16_DISP24
BFD_RELOC_CR16_DISP24a
BFD_RELOC_CR16_SWITCH8
BFD_RELOC_CR16_SWITCH16
BFD_RELOC_CR16_SWITCH32
BFD_RELOC_CR16_GOT_REGREL20
BFD_RELOC_CR16_GOTC_REGREL20
BFD_RELOC_CR16_GLOB_DAT
 NS CR16 Relocations.

BFD_RELOC_CRX_REL4
BFD_RELOC_CRX_REL8
BFD_RELOC_CRX_REL8_CMP
BFD_RELOC_CRX_REL16
BFD_RELOC_CRX_REL24
BFD_RELOC_CRX_REL32
BFD_RELOC_CRX_REGREL12
BFD_RELOC_CRX_REGREL22
BFD_RELOC_CRX_REGREL28
BFD_RELOC_CRX_REGREL32
BFD_RELOC_CRX_ABS16
BFD_RELOC_CRX_ABS32
BFD_RELOC_CRX_NUM8
BFD_RELOC_CRX_NUM16
BFD_RELOC_CRX_NUM32
BFD_RELOC_CRX_IMM16
BFD_RELOC_CRX_IMM32
BFD_RELOC_CRX_SWITCH8
BFD_RELOC_CRX_SWITCH16
BFD_RELOC_CRX_SWITCH32
 NS CRX Relocations.

BFD_RELOC_CRIS_BDISP8
BFD_RELOC_CRIS_UNSIGNED_5
BFD_RELOC_CRIS_SIGNED_6
BFD_RELOC_CRIS_UNSIGNED_6
BFD_RELOC_CRIS_SIGNED_8
BFD_RELOC_CRIS_UNSIGNED_8
BFD_RELOC_CRIS_SIGNED_16
BFD_RELOC_CRIS_UNSIGNED_16
BFD_RELOC_CRIS_LAPCQ_OFFSET
BFD_RELOC_CRIS_UNSIGNED_4

 These relocs are only used within the CRIS assembler. They are not (at present) written to any object files.

BFD_RELOC_CRIS_COPY
BFD_RELOC_CRIS_GLOB_DAT
BFD_RELOC_CRIS_JUMP_SLOT

BFD_RELOC_CRIS_RELATIVE

Relocs used in ELF shared libraries for CRIS.

BFD_RELOC_CRIS_32_GOT

32-bit offset to symbol-entry within GOT.

BFD_RELOC_CRIS_16_GOT

16-bit offset to symbol-entry within GOT.

BFD_RELOC_CRIS_32_GOTPLT

32-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_CRIS_16_GOTPLT

16-bit offset to symbol-entry within GOT, with PLT handling.

BFD_RELOC_CRIS_32_GOTREL

32-bit offset to symbol, relative to GOT.

BFD_RELOC_CRIS_32_PLT_GOTREL

32-bit offset to symbol with PLT entry, relative to GOT.

BFD_RELOC_CRIS_32_PLT_PCREL

32-bit offset to symbol with PLT entry, relative to this relocation.

BFD_RELOC_CRIS_32_GOT_GD

BFD_RELOC_CRIS_16_GOT_GD

BFD_RELOC_CRIS_32_GD

BFD_RELOC_CRIS_DTP

BFD_RELOC_CRIS_32_DTPREL

BFD_RELOC_CRIS_16_DTPREL

BFD_RELOC_CRIS_32_GOT_TPREL

BFD_RELOC_CRIS_16_GOT_TPREL

BFD_RELOC_CRIS_32_TPREL

BFD_RELOC_CRIS_16_TPREL

BFD_RELOC_CRIS_DTPMOD

BFD_RELOC_CRIS_32_IE

Relocs used in TLS code for CRIS.

BFD_RELOC_OR1K_REL_26

BFD_RELOC_OR1K_SL016

BFD_RELOC_OR1K_PCREL_PG21

BFD_RELOC_OR1K_L013

BFD_RELOC_OR1K_SL013

BFD_RELOC_OR1K_GOTPC_HI16

BFD_RELOC_OR1K_GOTPC_L016

BFD_RELOC_OR1K_GOT_AHI16

BFD_RELOC_OR1K_GOT16

BFD_RELOC_OR1K_GOT_PG21

BFD_RELOC_OR1K_GOT_L013

BFD_RELOC_OR1K_PLT26

BFD_RELOC_OR1K_PLTA26
BFD_RELOC_OR1K_GOTOFF_SLO16
BFD_RELOC_OR1K_COPY
BFD_RELOC_OR1K_GLOB_DAT
BFD_RELOC_OR1K_JMP_SLOT
BFD_RELOC_OR1K_RELATIVE
BFD_RELOC_OR1K_TLS_GD_HI16
BFD_RELOC_OR1K_TLS_GD_LO16
BFD_RELOC_OR1K_TLS_GD_PG21
BFD_RELOC_OR1K_TLS_GD_LO13
BFD_RELOC_OR1K_TLS_LDM_HI16
BFD_RELOC_OR1K_TLS_LDM_LO16
BFD_RELOC_OR1K_TLS_LDM_PG21
BFD_RELOC_OR1K_TLS_LDM_LO13
BFD_RELOC_OR1K_TLS_LDO_HI16
BFD_RELOC_OR1K_TLS_LDO_LO16
BFD_RELOC_OR1K_TLS_IE_HI16
BFD_RELOC_OR1K_TLS_IE_AHI16
BFD_RELOC_OR1K_TLS_IE_LO16
BFD_RELOC_OR1K_TLS_IE_PG21
BFD_RELOC_OR1K_TLS_IE_LO13
BFD_RELOC_OR1K_TLS_LE_HI16
BFD_RELOC_OR1K_TLS_LE_AHI16
BFD_RELOC_OR1K_TLS_LE_LO16
BFD_RELOC_OR1K_TLS_LE_SLO16
BFD_RELOC_OR1K_TLS_TPOFF
BFD_RELOC_OR1K_TLS_DTPOFF
BFD_RELOC_OR1K_TLS_DTPMOD

OpenRISC 1000 Relocations.

BFD_RELOC_H8_DIR16A8
BFD_RELOC_H8_DIR16R8
BFD_RELOC_H8_DIR24A8
BFD_RELOC_H8_DIR24R8
BFD_RELOC_H8_DIR32A16
BFD_RELOC_H8_DISP32A16

H8 elf Relocations.

BFD_RELOC_XSTORMY16_REL_12
BFD_RELOC_XSTORMY16_12
BFD_RELOC_XSTORMY16_24
BFD_RELOC_XSTORMY16_FPTR16

Sony Xstormy16 Relocations.

BFD_RELOC_RELC

Self-describing complex relocations.

BFD_RELOC_VAX_GLOB_DAT
BFD_RELOC_VAX_JMP_SLOT

BFD_RELOC_VAX_RELATIVE

Relocations used by VAX ELF.

BFD_RELOC_MT_PC16

Morpho MT - 16 bit immediate relocation.

BFD_RELOC_MT_HI16

Morpho MT - Hi 16 bits of an address.

BFD_RELOC_MT_LO16

Morpho MT - Low 16 bits of an address.

BFD_RELOC_MT_GNU_VTINHERIT

Morpho MT - Used to tell the linker which vtable entries are used.

BFD_RELOC_MT_GNU_VTENTRY

Morpho MT - Used to tell the linker which vtable entries are used.

BFD_RELOC_MT_PCINSN8

Morpho MT - 8 bit immediate relocation.

BFD_RELOC_MSP430_10_PCREL**BFD_RELOC_MSP430_16_PCREL****BFD_RELOC_MSP430_16****BFD_RELOC_MSP430_16_PCREL_BYTE****BFD_RELOC_MSP430_16_BYTE****BFD_RELOC_MSP430_2X_PCREL****BFD_RELOC_MSP430_RL_PCREL****BFD_RELOC_MSP430_ABS8****BFD_RELOC_MSP430X_PCR20_EXT_SRC****BFD_RELOC_MSP430X_PCR20_EXT_DST****BFD_RELOC_MSP430X_PCR20_EXT_ODST****BFD_RELOC_MSP430X_ABS20_EXT_SRC****BFD_RELOC_MSP430X_ABS20_EXT_DST****BFD_RELOC_MSP430X_ABS20_EXT_ODST****BFD_RELOC_MSP430X_ABS20_ADR_SRC****BFD_RELOC_MSP430X_ABS20_ADR_DST****BFD_RELOC_MSP430X_PCR16****BFD_RELOC_MSP430X_PCR20_CALL****BFD_RELOC_MSP430X_ABS16****BFD_RELOC_MSP430_ABS_HI16****BFD_RELOC_MSP430_PREL31****BFD_RELOC_MSP430_SYM_DIFF****BFD_RELOC_MSP430_SET_ULEB128****BFD_RELOC_MSP430_SUB_ULEB128**

msp430 specific relocation codes

BFD_RELOC_NIOS2_S16**BFD_RELOC_NIOS2_U16****BFD_RELOC_NIOS2_CALL26**

BFD_RELOC_NIOS2_IMM5
BFD_RELOC_NIOS2_CACHE_OPX
BFD_RELOC_NIOS2_IMM6
BFD_RELOC_NIOS2_IMM8
BFD_RELOC_NIOS2_HI16
BFD_RELOC_NIOS2_LO16
BFD_RELOC_NIOS2_HIADJ16
BFD_RELOC_NIOS2_GPREL
BFD_RELOC_NIOS2_UJMP
BFD_RELOC_NIOS2_CJMP
BFD_RELOC_NIOS2_CALLR
BFD_RELOC_NIOS2_ALIGN
BFD_RELOC_NIOS2_GOT16
BFD_RELOC_NIOS2_CALL16
BFD_RELOC_NIOS2_GOTOFF_LO
BFD_RELOC_NIOS2_GOTOFF_HA
BFD_RELOC_NIOS2_PCREL_LO
BFD_RELOC_NIOS2_PCREL_HA
BFD_RELOC_NIOS2_TLS_GD16
BFD_RELOC_NIOS2_TLS_LDM16
BFD_RELOC_NIOS2_TLS_LD016
BFD_RELOC_NIOS2_TLS_IE16
BFD_RELOC_NIOS2_TLS_LE16
BFD_RELOC_NIOS2_TLS_DTPMOD
BFD_RELOC_NIOS2_TLS_DTPREL
BFD_RELOC_NIOS2_TLS_TPREL
BFD_RELOC_NIOS2_COPY
BFD_RELOC_NIOS2_GLOB_DAT
BFD_RELOC_NIOS2_JUMP_SLOT
BFD_RELOC_NIOS2_RELATIVE
BFD_RELOC_NIOS2_GOTOFF
BFD_RELOC_NIOS2_CALL26_NOAT
BFD_RELOC_NIOS2_GOT_LO
BFD_RELOC_NIOS2_GOT_HA
BFD_RELOC_NIOS2_CALL_LO
BFD_RELOC_NIOS2_CALL_HA
BFD_RELOC_NIOS2_R2_S12
BFD_RELOC_NIOS2_R2_I10_1_PCREL
BFD_RELOC_NIOS2_R2_T1I7_1_PCREL
BFD_RELOC_NIOS2_R2_T1I7_2
BFD_RELOC_NIOS2_R2_T2I4
BFD_RELOC_NIOS2_R2_T2I4_1
BFD_RELOC_NIOS2_R2_T2I4_2
BFD_RELOC_NIOS2_R2_X1I7_2
BFD_RELOC_NIOS2_R2_X2L5
BFD_RELOC_NIOS2_R2_F1I5_2
BFD_RELOC_NIOS2_R2_L5I4X1

BFD_RELOC_NIOS2_R2_T1X1I6

BFD_RELOC_NIOS2_R2_T1X1I6_2

Relocations used by the Altera Nios II core.

BFD_RELOC_PRU_U16

PRU LDI 16-bit unsigned data-memory relocation.

BFD_RELOC_PRU_U16_PMEMIMM

PRU LDI 16-bit unsigned instruction-memory relocation.

BFD_RELOC_PRU_LDI32

PRU relocation for two consecutive LDI load instructions that load a 32 bit value into a register. If the higher bits are all zero, then the second instruction may be relaxed.

BFD_RELOC_PRU_S10_PCREL

PRU QBBx 10-bit signed PC-relative relocation.

BFD_RELOC_PRU_U8_PCREL

PRU 8-bit unsigned relocation used for the LOOP instruction.

BFD_RELOC_PRU_32_PMEM

BFD_RELOC_PRU_16_PMEM

PRU Program Memory relocations. Used to convert from byte addressing to 32-bit word addressing.

BFD_RELOC_PRU_GNU_DIFF8

BFD_RELOC_PRU_GNU_DIFF16

BFD_RELOC_PRU_GNU_DIFF32

BFD_RELOC_PRU_GNU_DIFF16_PMEM

BFD_RELOC_PRU_GNU_DIFF32_PMEM

PRU relocations to mark the difference of two local symbols. These are only needed to support linker relaxation and can be ignored when not relaxing. The field is set to the value of the difference assuming no relaxation. The relocation encodes the position of the second symbol so the linker can determine whether to adjust the field value. The PMEM variants encode the word difference, instead of byte difference between symbols.

BFD_RELOC_IQ2000_OFFSET_16

BFD_RELOC_IQ2000_OFFSET_21

BFD_RELOC_IQ2000_UHI16

IQ2000 Relocations.

BFD_RELOC_XTENSA_RTLD

Special Xtensa relocation used only by PLT entries in ELF shared objects to indicate that the runtime linker should set the value to one of its own internal functions or data structures.

BFD_RELOC_XTENSA_GLOB_DAT

BFD_RELOC_XTENSA_JMP_SLOT

BFD_RELOC_XTENSA_RELATIVE

Xtensa relocations for ELF shared objects.

BFD_RELOC_XTENSA_PLT

Xtensa relocation used in ELF object files for symbols that may require PLT entries. Otherwise, this is just a generic 32-bit relocation.

BFD_RELOC_XTENSA_DIFF8**BFD_RELOC_XTENSA_DIFF16****BFD_RELOC_XTENSA_DIFF32**

Xtensa relocations for backward compatibility. These have been replaced by **BFD_RELOC_XTENSA_PDIFF** and **BFD_RELOC_XTENSA_NDIFF**. Xtensa relocations to mark the difference of two local symbols. These are only needed to support linker relaxation and can be ignored when not relaxing. The field is set to the value of the difference assuming no relaxation. The relocation encodes the position of the first symbol so the linker can determine whether to adjust the field value.

BFD_RELOC_XTENSA_SLOT0_OP**BFD_RELOC_XTENSA_SLOT1_OP****BFD_RELOC_XTENSA_SLOT2_OP****BFD_RELOC_XTENSA_SLOT3_OP****BFD_RELOC_XTENSA_SLOT4_OP****BFD_RELOC_XTENSA_SLOT5_OP****BFD_RELOC_XTENSA_SLOT6_OP****BFD_RELOC_XTENSA_SLOT7_OP****BFD_RELOC_XTENSA_SLOT8_OP****BFD_RELOC_XTENSA_SLOT9_OP****BFD_RELOC_XTENSA_SLOT10_OP****BFD_RELOC_XTENSA_SLOT11_OP****BFD_RELOC_XTENSA_SLOT12_OP****BFD_RELOC_XTENSA_SLOT13_OP****BFD_RELOC_XTENSA_SLOT14_OP**

Generic Xtensa relocations for instruction operands. Only the slot number is encoded in the relocation. The relocation applies to the last PC-relative immediate operand, or if there are no PC-relative immediates, to the last immediate operand.

BFD_RELOC_XTENSA_SLOT0_ALT**BFD_RELOC_XTENSA_SLOT1_ALT****BFD_RELOC_XTENSA_SLOT2_ALT****BFD_RELOC_XTENSA_SLOT3_ALT****BFD_RELOC_XTENSA_SLOT4_ALT****BFD_RELOC_XTENSA_SLOT5_ALT****BFD_RELOC_XTENSA_SLOT6_ALT****BFD_RELOC_XTENSA_SLOT7_ALT****BFD_RELOC_XTENSA_SLOT8_ALT****BFD_RELOC_XTENSA_SLOT9_ALT****BFD_RELOC_XTENSA_SLOT10_ALT****BFD_RELOC_XTENSA_SLOT11_ALT****BFD_RELOC_XTENSA_SLOT12_ALT****BFD_RELOC_XTENSA_SLOT13_ALT**

BFD_RELOC_XTENSA_SLOT14_ALT

Alternate Xtensa relocations. Only the slot is encoded in the relocation. The meaning of these relocations is opcode-specific.

BFD_RELOC_XTENSA_OPO**BFD_RELOC_XTENSA_OP1****BFD_RELOC_XTENSA_OP2**

Xtensa relocations for backward compatibility. These have all been replaced by BFD_RELOC_XTENSA_SLOT0_OP.

BFD_RELOC_XTENSA_ASM_EXPAND

Xtensa relocation to mark that the assembler expanded the instructions from an original target. The expansion size is encoded in the reloc size.

BFD_RELOC_XTENSA_ASM_SIMPLIFY

Xtensa relocation to mark that the linker should simplify assembler-expanded instructions. This is commonly used internally by the linker after analysis of a BFD_RELOC_XTENSA_ASM_EXPAND.

BFD_RELOC_XTENSA_TLSDESC_FN**BFD_RELOC_XTENSA_TLSDESC_ARG****BFD_RELOC_XTENSA_TLS_DTPOFF****BFD_RELOC_XTENSA_TLS_TPOFF****BFD_RELOC_XTENSA_TLS_FUNC****BFD_RELOC_XTENSA_TLS_ARG****BFD_RELOC_XTENSA_TLS_CALL**

Xtensa TLS relocations.

BFD_RELOC_XTENSA_PDIFF8**BFD_RELOC_XTENSA_PDIFF16****BFD_RELOC_XTENSA_PDIFF32****BFD_RELOC_XTENSA_NDIFF8****BFD_RELOC_XTENSA_NDIFF16****BFD_RELOC_XTENSA_NDIFF32**

Xtensa relocations to mark the difference of two local symbols. These are only needed to support linker relaxation and can be ignored when not relaxing. The field is set to the value of the difference assuming no relaxation. The relocation encodes the position of the subtracted symbol so the linker can determine whether to adjust the field value. PDIFF relocations are used for positive differences, NDIFF relocations are used for negative differences. The difference value is treated as unsigned with these relocation types, giving full 8/16 value ranges.

BFD_RELOC_Z80_DISP8

8 bit signed offset in (ix+d) or (iy+d).

BFD_RELOC_Z80_BYTE0

First 8 bits of multibyte (32, 24 or 16 bit) value.

BFD_RELOC_Z80_BYTE1

Second 8 bits of multibyte (32, 24 or 16 bit) value.

BFD_RELOC_Z80_BYTE2

Third 8 bits of multibyte (32 or 24 bit) value.

BFD_RELOC_Z80_BYTE3

Fourth 8 bits of multibyte (32 bit) value.

BFD_RELOC_Z80_WORD0

Lowest 16 bits of multibyte (32 or 24 bit) value.

BFD_RELOC_Z80_WORD1

Highest 16 bits of multibyte (32 or 24 bit) value.

BFD_RELOC_Z80_16_BE

Like BFD_RELOC_16 but big-endian.

BFD_RELOC_Z8K_DISP7

DJNZ offset.

BFD_RELOC_Z8K_CALLR

CALR offset.

BFD_RELOC_Z8K_IMM4L

4 bit value.

BFD_RELOC_LM32_CALL

BFD_RELOC_LM32_BRANCH

BFD_RELOC_LM32_16_GOT

BFD_RELOC_LM32_GOTOFF_HI16

BFD_RELOC_LM32_GOTOFF_LO16

BFD_RELOC_LM32_COPY

BFD_RELOC_LM32_GLOB_DAT

BFD_RELOC_LM32_JMP_SLOT

BFD_RELOC_LM32_RELATIVE

Lattice Mico32 relocations.

BFD_RELOC_MACH_O_SECTDIFF

Difference between two section addresses. Must be followed by a BFD_RELOC_MACH_O_PAIR.

BFD_RELOC_MACH_O_LOCAL_SECTDIFF

Like BFD_RELOC_MACH_O_SECTDIFF but with a local symbol.

BFD_RELOC_MACH_O_PAIR

Pair of relocation. Contains the first symbol.

BFD_RELOC_MACH_O_SUBTRACTOR32

Symbol will be subtracted. Must be followed by a BFD_RELOC_32.

BFD_RELOC_MACH_O_SUBTRACTOR64

Symbol will be subtracted. Must be followed by a BFD_RELOC_64.

BFD_RELOC_MACH_O_X86_64_BRANCH32

BFD_RELOC_MACH_O_X86_64_BRANCH8

PCREL relocations. They are marked as branch to create PLT entry if required.

BFD_RELOC_MACH_O_X86_64_GOT

Used when referencing a GOT entry.

BFD_RELOC_MACH_O_X86_64_GOT_LOAD

Used when loading a GOT entry with movq. It is specially marked so that the linker could optimize the movq to a leaq if possible.

BFD_RELOC_MACH_O_X86_64_PCREL32_1

Same as BFD_RELOC_32_PCREL but with an implicit -1 addend.

BFD_RELOC_MACH_O_X86_64_PCREL32_2

Same as BFD_RELOC_32_PCREL but with an implicit -2 addend.

BFD_RELOC_MACH_O_X86_64_PCREL32_4

Same as BFD_RELOC_32_PCREL but with an implicit -4 addend.

BFD_RELOC_MACH_O_X86_64_TLV

Used when referencing a TLV entry.

BFD_RELOC_MACH_O_ARM64_ADDEND

Addend for PAGE or PAGEOFF.

BFD_RELOC_MACH_O_ARM64_GOT_LOAD_PAGE21

Relative offset to page of GOT slot.

BFD_RELOC_MACH_O_ARM64_GOT_LOAD_PAGEOFF12

Relative offset within page of GOT slot.

BFD_RELOC_MACH_O_ARM64_POINTER_TO_GOT

Address of a GOT entry.

BFD_RELOC_MICROBLAZE_32_LO

This is a 32 bit reloc for the microblaze that stores the low 16 bits of a value

BFD_RELOC_MICROBLAZE_32_LO_PCREL

This is a 32 bit pc-relative reloc for the microblaze that stores the low 16 bits of a value

BFD_RELOC_MICROBLAZE_32_ROSDA

This is a 32 bit reloc for the microblaze that stores a value relative to the read-only small data area anchor

BFD_RELOC_MICROBLAZE_32_RWSDA

This is a 32 bit reloc for the microblaze that stores a value relative to the read-write small data area anchor

BFD_RELOC_MICROBLAZE_32_SYM_OP_SYM

This is a 32 bit reloc for the microblaze to handle expressions of the form "Symbol Op Symbol"

BFD_RELOC_MICROBLAZE_64_NONE

This is a 64 bit reloc that stores the 32 bit pc relative value in two words (with an imm instruction). No relocation is done here - only used for relaxing

BFD_RELOC_MICROBLAZE_64_GOTPC

This is a 64 bit reloc that stores the 32 bit pc relative value in two words (with an imm instruction). The relocation is PC-relative GOT offset

BFD_RELOC_MICROBLAZE_64_GOT

This is a 64 bit reloc that stores the 32 bit pc relative value in two words (with an imm instruction). The relocation is GOT offset

BFD_RELOC_MICROBLAZE_64_PLT

This is a 64 bit reloc that stores the 32 bit pc relative value in two words (with an imm instruction). The relocation is PC-relative offset into PLT

BFD_RELOC_MICROBLAZE_64_GOTOFF

This is a 64 bit reloc that stores the 32 bit GOT relative value in two words (with an imm instruction). The relocation is relative offset from `_GLOBAL_OFFSET_TABLE_`

BFD_RELOC_MICROBLAZE_32_GOTOFF

This is a 32 bit reloc that stores the 32 bit GOT relative value in a word. The relocation is relative offset from

BFD_RELOC_MICROBLAZE_COPY

This is used to tell the dynamic linker to copy the value out of the dynamic object into the runtime process image.

BFD_RELOC_MICROBLAZE_64_TLS

Unused Reloc

BFD_RELOC_MICROBLAZE_64_TLSD

This is a 64 bit reloc that stores the 32 bit GOT relative value of the GOT TLS GD info entry in two words (with an imm instruction). The relocation is GOT offset.

BFD_RELOC_MICROBLAZE_64_TLSD

This is a 64 bit reloc that stores the 32 bit GOT relative value of the GOT TLS LD info entry in two words (with an imm instruction). The relocation is GOT offset.

BFD_RELOC_MICROBLAZE_32_TLSDTPMOD

This is a 32 bit reloc that stores the Module ID to GOT(n).

BFD_RELOC_MICROBLAZE_32_TLSDTPREL

This is a 32 bit reloc that stores TLS offset to GOT(n+1).

BFD_RELOC_MICROBLAZE_64_TLSDTPREL

This is a 32 bit reloc for storing TLS offset to two words (uses imm instruction)

BFD_RELOC_MICROBLAZE_64_TLSTP

This is a 64 bit reloc that stores 32-bit thread pointer relative offset to two words (uses imm instruction).

BFD_RELOC_MICROBLAZE_64_TLSTPREL

This is a 64 bit reloc that stores 32-bit thread pointer relative offset to two words (uses imm instruction).

BFD_RELOC_MICROBLAZE_64_TEXTPCREL

This is a 64 bit reloc that stores the 32 bit pc relative value in two words (with an imm instruction). The relocation is PC-relative offset from start of TEXT.

BFD_RELOC_MICROBLAZE_64_TEXTREL

This is a 64 bit reloc that stores the 32 bit offset value in two words (with an imm instruction). The relocation is relative offset from start of TEXT.

BFD_RELOC_AARCH64_RELOC_START

AArch64 pseudo relocation code to mark the start of the AArch64 relocation enumerators. N.B. the order of the enumerators is important as several tables in the AArch64 bfd backend are indexed by these enumerators; make sure they are all synced.

BFD_RELOC_AARCH64_NULL

Deprecated AArch64 null relocation code.

BFD_RELOC_AARCH64_NONE

AArch64 null relocation code.

BFD_RELOC_AARCH64_64**BFD_RELOC_AARCH64_32****BFD_RELOC_AARCH64_16**

Basic absolute relocations of N bits. These are equivalent to BFD_RELOC_N and they were added to assist the indexing of the howto table.

BFD_RELOC_AARCH64_64_PCREL**BFD_RELOC_AARCH64_32_PCREL****BFD_RELOC_AARCH64_16_PCREL**

PC-relative relocations. These are equivalent to BFD_RELOC_N_PCREL and they were added to assist the indexing of the howto table.

BFD_RELOC_AARCH64_MOVW_GO

AArch64 MOV[NZK] instruction with most significant bits 0 to 15 of an unsigned address/value.

BFD_RELOC_AARCH64_MOVW_GO_NC

AArch64 MOV[NZK] instruction with less significant bits 0 to 15 of an address/value. No overflow checking.

BFD_RELOC_AARCH64_MOVW_G1

AArch64 MOV[NZK] instruction with most significant bits 16 to 31 of an unsigned address/value.

BFD_RELOC_AARCH64_MOVW_G1_NC

AArch64 MOV[NZK] instruction with less significant bits 16 to 31 of an address/value. No overflow checking.

BFD_RELOC_AARCH64_MOVW_G2

AArch64 MOV[NZK] instruction with most significant bits 32 to 47 of an unsigned address/value.

BFD_RELOC_AARCH64_MOVW_G2_NC

AArch64 MOV[NZK] instruction with less significant bits 32 to 47 of an address/value. No overflow checking.

BFD_RELOC_AARCH64_MOVW_G3

AArch64 MOV[NZK] instruction with most significant bits 48 to 64 of a signed or unsigned address/value.

BFD_RELOC_AARCH64_MOVW_G0_S

AArch64 MOV[NZ] instruction with most significant bits 0 to 15 of a signed value. Changes instruction to MOVZ or MOVN depending on the value's sign.

BFD_RELOC_AARCH64_MOVW_G1_S

AArch64 MOV[NZ] instruction with most significant bits 16 to 31 of a signed value. Changes instruction to MOVZ or MOVN depending on the value's sign.

BFD_RELOC_AARCH64_MOVW_G2_S

AArch64 MOV[NZ] instruction with most significant bits 32 to 47 of a signed value. Changes instruction to MOVZ or MOVN depending on the value's sign.

BFD_RELOC_AARCH64_MOVW_PREL_G0

AArch64 MOV[NZ] instruction with most significant bits 0 to 15 of a signed value. Changes instruction to MOVZ or MOVN depending on the value's sign.

BFD_RELOC_AARCH64_MOVW_PREL_G0_NC

AArch64 MOV[NZ] instruction with most significant bits 0 to 15 of a signed value. Changes instruction to MOVZ or MOVN depending on the value's sign.

BFD_RELOC_AARCH64_MOVW_PREL_G1

AArch64 MOVK instruction with most significant bits 16 to 31 of a signed value.

BFD_RELOC_AARCH64_MOVW_PREL_G1_NC

AArch64 MOVK instruction with most significant bits 16 to 31 of a signed value.

BFD_RELOC_AARCH64_MOVW_PREL_G2

AArch64 MOVK instruction with most significant bits 32 to 47 of a signed value.

BFD_RELOC_AARCH64_MOVW_PREL_G2_NC

AArch64 MOVK instruction with most significant bits 32 to 47 of a signed value.

BFD_RELOC_AARCH64_MOVW_PREL_G3

AArch64 MOVK instruction with most significant bits 47 to 63 of a signed value.

BFD_RELOC_AARCH64_LD_L019_PCREL

AArch64 Load Literal instruction, holding a 19 bit pc-relative word offset. The lowest two bits must be zero and are not stored in the instruction, giving a 21 bit signed byte offset.

BFD_RELOC_AARCH64_ADR_L021_PCREL

AArch64 ADR instruction, holding a simple 21 bit pc-relative byte offset.

BFD_RELOC_AARCH64_ADR_HI21_PCREL

AArch64 ADRP instruction, with bits 12 to 32 of a pc-relative page offset, giving a 4KB aligned page base address.

BFD_RELOC_AARCH64_ADR_HI21_NC_PCREL

AArch64 ADRP instruction, with bits 12 to 32 of a pc-relative page offset, giving a 4KB aligned page base address, but with no overflow checking.

BFD_RELOC_AARCH64_ADD_L012

AArch64 ADD immediate instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_LDST8_L012

AArch64 8-bit load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_TSTBR14

AArch64 14 bit pc-relative test bit and branch. The lowest two bits must be zero and are not stored in the instruction, giving a 16 bit signed byte offset.

BFD_RELOC_AARCH64_BRANCH19

AArch64 19 bit pc-relative conditional branch and compare & branch. The lowest two bits must be zero and are not stored in the instruction, giving a 21 bit signed byte offset.

BFD_RELOC_AARCH64_JUMP26

AArch64 26 bit pc-relative unconditional branch. The lowest two bits must be zero and are not stored in the instruction, giving a 28 bit signed byte offset.

BFD_RELOC_AARCH64_CALL26

AArch64 26 bit pc-relative unconditional branch and link. The lowest two bits must be zero and are not stored in the instruction, giving a 28 bit signed byte offset.

BFD_RELOC_AARCH64_LDST16_L012

AArch64 16-bit load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_LDST32_L012

AArch64 32-bit load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_LDST64_L012

AArch64 64-bit load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_LDST128_L012

AArch64 128-bit load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_GOT_LD_PREL19

AArch64 Load Literal instruction, holding a 19 bit PC relative word offset of the global offset table entry for a symbol. The lowest two bits must be zero and are not stored in the instruction, giving a 21 bit signed byte offset. This relocation type requires signed overflow checking.

BFD_RELOC_AARCH64_ADR_GOT_PAGE

Get to the page base of the global offset table entry for a symbol as part of an ADRP instruction using a 21 bit PC relative value. Used in conjunction with BFD_RELOC_AARCH64_LD64_GOT_LO12_NC.

BFD_RELOC_AARCH64_LD64_GOT_LO12_NC

Unsigned 12 bit byte offset for 64 bit load/store from the page of the GOT entry for this symbol. Used in conjunction with BFD_RELOC_AARCH64_ADR_GOT_PAGE. Valid in LP64 ABI only.

BFD_RELOC_AARCH64_LD32_GOT_LO12_NC

Unsigned 12 bit byte offset for 32 bit load/store from the page of the GOT entry for this symbol. Used in conjunction with BFD_RELOC_AARCH64_ADR_GOT_PAGE. Valid in ILP32 ABI only.

BFD_RELOC_AARCH64_MOVW_GOTOFF_GO_NC

Unsigned 16 bit byte offset for 64 bit load/store from the GOT entry for this symbol. Valid in LP64 ABI only.

BFD_RELOC_AARCH64_MOVW_GOTOFF_G1

Unsigned 16 bit byte higher offset for 64 bit load/store from the GOT entry for this symbol. Valid in LP64 ABI only.

BFD_RELOC_AARCH64_LD64_GOTOFF_LO15

Unsigned 15 bit byte offset for 64 bit load/store from the page of the GOT entry for this symbol. Valid in LP64 ABI only.

BFD_RELOC_AARCH64_LD32_GOTPAGE_LO14

Scaled 14 bit byte offset to the page base of the global offset table.

BFD_RELOC_AARCH64_LD64_GOTPAGE_LO15

Scaled 15 bit byte offset to the page base of the global offset table.

BFD_RELOC_AARCH64_TLSGD_ADR_PAGE21

Get to the page base of the global offset table entry for a symbols `tls_index` structure as part of an `adrp` instruction using a 21 bit PC relative value. Used in conjunction with BFD_RELOC_AARCH64_TLSGD_ADD_LO12_NC.

BFD_RELOC_AARCH64_TLSGD_ADR_PREL21

AArch64 TLS General Dynamic

BFD_RELOC_AARCH64_TLSGD_ADD_LO12_NC

Unsigned 12 bit byte offset to global offset table entry for a symbols `tls_index` structure. Used in conjunction with BFD_RELOC_AARCH64_TLSGD_ADR_PAGE21.

- BFD_RELOC_AARCH64_TLSGD_MOVW_GO_NC**
AArch64 TLS General Dynamic relocation.
- BFD_RELOC_AARCH64_TLSGD_MOVW_G1**
AArch64 TLS General Dynamic relocation.
- BFD_RELOC_AARCH64_TLSIE_ADR_GOTTPREL_PAGE21**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSIE_LD64_GOTTPREL_LO12_NC**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSIE_LD32_GOTTPREL_LO12_NC**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSIE_LD_GOTTPREL_PREL19**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSIE_MOVW_GOTTPREL_GO_NC**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSIE_MOVW_GOTTPREL_G1**
AArch64 TLS INITIAL EXEC relocation.
- BFD_RELOC_AARCH64_TLSLD_ADD_DTPREL_HI12**
bit[23:12] of byte offset to module TLS base address.
- BFD_RELOC_AARCH64_TLSLD_ADD_DTPREL_LO12**
Unsigned 12 bit byte offset to module TLS base address.
- BFD_RELOC_AARCH64_TLSLD_ADD_DTPREL_LO12_NC**
No overflow check version of BFD_RELOC_AARCH64_TLSLD_ADD_DTPREL_LO12.
- BFD_RELOC_AARCH64_TLSLD_ADD_LO12_NC**
Unsigned 12 bit byte offset to global offset table entry for a symbols tls_index structure.
Used in conjunction with BFD_RELOC_AARCH64_TLSLD_ADR_PAGE21.
- BFD_RELOC_AARCH64_TLSLD_ADR_PAGE21**
GOT entry page address for AArch64 TLS Local Dynamic, used with ADRP instruction.
- BFD_RELOC_AARCH64_TLSLD_ADR_PREL21**
GOT entry address for AArch64 TLS Local Dynamic, used with ADR instruction.
- BFD_RELOC_AARCH64_TLSLD_LDST16_DTPREL_LO12**
bit[11:1] of byte offset to module TLS base address, encoded in ldst instructions.
- BFD_RELOC_AARCH64_TLSLD_LDST16_DTPREL_LO12_NC**
Similar as BFD_RELOC_AARCH64_TLSLD_LDST16_DTPREL_LO12, but no overflow check.
- BFD_RELOC_AARCH64_TLSLD_LDST32_DTPREL_LO12**
bit[11:2] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLD_LDST32_DTPREL_L012_NC

Similar as BFD_RELOC_AARCH64_TLSLD_LDST32_DTPREL_LO12, but no overflow check.

BFD_RELOC_AARCH64_TLSLD_LDST64_DTPREL_L012

bit[11:3] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLD_LDST64_DTPREL_L012_NC

Similar as BFD_RELOC_AARCH64_TLSLD_LDST64_DTPREL_LO12, but no overflow check.

BFD_RELOC_AARCH64_TLSLD_LDST8_DTPREL_L012

bit[11:0] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLD_LDST8_DTPREL_L012_NC

Similar as BFD_RELOC_AARCH64_TLSLD_LDST8_DTPREL_LO12, but no overflow check.

BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G0

bit[15:0] of byte offset to module TLS base address.

BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G0_NC

No overflow check version of BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G0

BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G1

bit[31:16] of byte offset to module TLS base address.

BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G1_NC

No overflow check version of BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G1

BFD_RELOC_AARCH64_TLSLD_MOVW_DTPREL_G2

bit[47:32] of byte offset to module TLS base address.

BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G2

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G1

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G1_NC

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G0

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G0_NC

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_ADD_TPREL_HI12

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_ADD_TPREL_L012

AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_ADD_TPREL_L012_NC
AArch64 TLS LOCAL EXEC relocation.

BFD_RELOC_AARCH64_TLSLE_LDST16_TPREL_L012
bit[11:1] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLE_LDST16_TPREL_L012_NC
Similar as BFD_RELOC_AARCH64_TLSLE_LDST16_TPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_TLSLE_LDST32_TPREL_L012
bit[11:2] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLE_LDST32_TPREL_L012_NC
Similar as BFD_RELOC_AARCH64_TLSLE_LDST32_TPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_TLSLE_LDST64_TPREL_L012
bit[11:3] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLE_LDST64_TPREL_L012_NC
Similar as BFD_RELOC_AARCH64_TLSLE_LDST64_TPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_TLSLE_LDST8_TPREL_L012
bit[11:0] of byte offset to module TLS base address, encoded in ldst instructions.

BFD_RELOC_AARCH64_TLSLE_LDST8_TPREL_L012_NC
Similar as BFD_RELOC_AARCH64_TLSLE_LDST8_TPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_TLSDESC_LD_PREL19
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_ADR_PREL21
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_ADR_PAGE21
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_LD64_L012
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_LD32_L012_NC
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_ADD_L012
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_OFF_G1
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_OFF_GO_NC
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_LDR
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_ADD
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_TLSDESC_CALL
AArch64 TLS DESC relocation.

BFD_RELOC_AARCH64_COPY
AArch64 TLS relocation.

BFD_RELOC_AARCH64_GLOB_DAT
AArch64 TLS relocation.

BFD_RELOC_AARCH64_JUMP_SLOT
AArch64 TLS relocation.

BFD_RELOC_AARCH64_RELATIVE
AArch64 TLS relocation.

BFD_RELOC_AARCH64_TLS_DTPMOD
AArch64 TLS relocation.

BFD_RELOC_AARCH64_TLS_DTPREL
AArch64 TLS relocation.

BFD_RELOC_AARCH64_TLS_TPREL
AArch64 TLS relocation.

BFD_RELOC_AARCH64_TLSDESC
AArch64 TLS relocation.

BFD_RELOC_AARCH64_IRELATIVE
AArch64 support for STT_GNU_IFUNC.

BFD_RELOC_AARCH64_RELOC_END
AArch64 pseudo relocation code to mark the end of the AArch64 relocation enumerators that have direct mapping to ELF reloc codes. There are a few more enumerators after this one; those are mainly used by the AArch64 assembler for the internal fixup or to select one of the above enumerators.

BFD_RELOC_AARCH64_GAS_INTERNAL_FIXUP
AArch64 pseudo relocation code to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_AARCH64_LDST_L012
AArch64 unspecified load/store instruction, holding bits 0 to 11 of the address. Used in conjunction with BFD_RELOC_AARCH64_ADR_HI21_PCREL.

BFD_RELOC_AARCH64_TLSLD_LDST_DTPREL_L012

AArch64 pseudo relocation code for TLS local dynamic mode. It's to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_AARCH64_TLSLD_LDST_DTPREL_L012_NC

Similar as BFD_RELOC_AARCH64_TLSLD_LDST_DTPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_TLSLE_LDST_TPREL_L012

AArch64 pseudo relocation code for TLS local exec mode. It's to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_AARCH64_TLSLE_LDST_TPREL_L012_NC

Similar as BFD_RELOC_AARCH64_TLSLE_LDST_TPREL_L012, but no overflow check.

BFD_RELOC_AARCH64_LD_GOT_L012_NC

AArch64 pseudo relocation code to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_AARCH64_TLSIE_LD_GOTTPREL_L012_NC

AArch64 pseudo relocation code to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_AARCH64_TLSDESC_LD_L012_NC

AArch64 pseudo relocation code to be used internally by the AArch64 assembler and not (currently) written to any object files.

BFD_RELOC_TILEPRO_COPY**BFD_RELOC_TILEPRO_GLOB_DAT****BFD_RELOC_TILEPRO_JMP_SLOT****BFD_RELOC_TILEPRO_RELATIVE****BFD_RELOC_TILEPRO_BROFF_X1****BFD_RELOC_TILEPRO_JOFFLONG_X1****BFD_RELOC_TILEPRO_JOFFLONG_X1_PLT****BFD_RELOC_TILEPRO_IMM8_X0****BFD_RELOC_TILEPRO_IMM8_Y0****BFD_RELOC_TILEPRO_IMM8_X1****BFD_RELOC_TILEPRO_IMM8_Y1****BFD_RELOC_TILEPRO_DEST_IMM8_X1****BFD_RELOC_TILEPRO_MT_IMM15_X1****BFD_RELOC_TILEPRO_MF_IMM15_X1****BFD_RELOC_TILEPRO_IMM16_X0****BFD_RELOC_TILEPRO_IMM16_X1****BFD_RELOC_TILEPRO_IMM16_X0_LO****BFD_RELOC_TILEPRO_IMM16_X1_LO****BFD_RELOC_TILEPRO_IMM16_X0_HI****BFD_RELOC_TILEPRO_IMM16_X1_HI****BFD_RELOC_TILEPRO_IMM16_X0_HA**

BFD_RELOC_TILEPRO_IMM16_X1_HA
BFD_RELOC_TILEPRO_IMM16_X0_PCREL
BFD_RELOC_TILEPRO_IMM16_X1_PCREL
BFD_RELOC_TILEPRO_IMM16_X0_LO_PCREL
BFD_RELOC_TILEPRO_IMM16_X1_LO_PCREL
BFD_RELOC_TILEPRO_IMM16_X0_HI_PCREL
BFD_RELOC_TILEPRO_IMM16_X1_HI_PCREL
BFD_RELOC_TILEPRO_IMM16_X0_HA_PCREL
BFD_RELOC_TILEPRO_IMM16_X1_HA_PCREL
BFD_RELOC_TILEPRO_IMM16_X0_GOT
BFD_RELOC_TILEPRO_IMM16_X1_GOT
BFD_RELOC_TILEPRO_IMM16_X0_GOT_LO
BFD_RELOC_TILEPRO_IMM16_X1_GOT_LO
BFD_RELOC_TILEPRO_IMM16_X0_GOT_HI
BFD_RELOC_TILEPRO_IMM16_X1_GOT_HI
BFD_RELOC_TILEPRO_IMM16_X0_GOT_HA
BFD_RELOC_TILEPRO_IMM16_X1_GOT_HA
BFD_RELOC_TILEPRO_MMSTART_X0
BFD_RELOC_TILEPRO_MMEND_X0
BFD_RELOC_TILEPRO_MMSTART_X1
BFD_RELOC_TILEPRO_MMEND_X1
BFD_RELOC_TILEPRO_SHAMT_X0
BFD_RELOC_TILEPRO_SHAMT_X1
BFD_RELOC_TILEPRO_SHAMT_Y0
BFD_RELOC_TILEPRO_SHAMT_Y1
BFD_RELOC_TILEPRO_TLS_GD_CALL
BFD_RELOC_TILEPRO_IMM8_X0_TLS_GD_ADD
BFD_RELOC_TILEPRO_IMM8_X1_TLS_GD_ADD
BFD_RELOC_TILEPRO_IMM8_Y0_TLS_GD_ADD
BFD_RELOC_TILEPRO_IMM8_Y1_TLS_GD_ADD
BFD_RELOC_TILEPRO_TLS_IE_LOAD
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_LO
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_LO
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_HI
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_HI
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_HA
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_HA
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_LO
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_LO
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_HI
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_HI
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_HA
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_HA

BFD_RELOC_TILEPRO_TLS_DTPMOD32
BFD_RELOC_TILEPRO_TLS_DTPOFF32
BFD_RELOC_TILEPRO_TLS_TPOFF32
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_LO
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_LO
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_HI
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_HI
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_HA
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_HA

Tilera TILEPro Relocations.

BFD_RELOC_TILEGX_HWO
BFD_RELOC_TILEGX_HW1
BFD_RELOC_TILEGX_HW2
BFD_RELOC_TILEGX_HW3
BFD_RELOC_TILEGX_HWO_LAST
BFD_RELOC_TILEGX_HW1_LAST
BFD_RELOC_TILEGX_HW2_LAST
BFD_RELOC_TILEGX_COPY
BFD_RELOC_TILEGX_GLOB_DAT
BFD_RELOC_TILEGX_JMP_SLOT
BFD_RELOC_TILEGX_RELATIVE
BFD_RELOC_TILEGX_BROFF_X1
BFD_RELOC_TILEGX_JUMPOFF_X1
BFD_RELOC_TILEGX_JUMPOFF_X1_PLT
BFD_RELOC_TILEGX_IMM8_X0
BFD_RELOC_TILEGX_IMM8_Y0
BFD_RELOC_TILEGX_IMM8_X1
BFD_RELOC_TILEGX_IMM8_Y1
BFD_RELOC_TILEGX_DEST_IMM8_X1
BFD_RELOC_TILEGX_MT_IMM14_X1
BFD_RELOC_TILEGX_MF_IMM14_X1
BFD_RELOC_TILEGX_MMSTART_X0
BFD_RELOC_TILEGX_MMEND_X0
BFD_RELOC_TILEGX_SHAMT_X0
BFD_RELOC_TILEGX_SHAMT_X1
BFD_RELOC_TILEGX_SHAMT_Y0
BFD_RELOC_TILEGX_SHAMT_Y1
BFD_RELOC_TILEGX_IMM16_X0_HWO
BFD_RELOC_TILEGX_IMM16_X1_HWO
BFD_RELOC_TILEGX_IMM16_X0_HW1
BFD_RELOC_TILEGX_IMM16_X1_HW1
BFD_RELOC_TILEGX_IMM16_X0_HW2
BFD_RELOC_TILEGX_IMM16_X1_HW2
BFD_RELOC_TILEGX_IMM16_X0_HW3

BFD_RELOC_TILEGX_IMM16_X1_HW3
BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST
BFD_RELOC_TILEGX_IMM16_X0_HW2_LAST
BFD_RELOC_TILEGX_IMM16_X1_HW2_LAST
BFD_RELOC_TILEGX_IMM16_X0_HWO_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HWO_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW1_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW1_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW2_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW2_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW3_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW3_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW2_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW2_LAST_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HWO_GOT
BFD_RELOC_TILEGX_IMM16_X1_HWO_GOT
BFD_RELOC_TILEGX_IMM16_X0_HWO_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HWO_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW1_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW1_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HW2_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW2_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_GOT
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_GOT
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_GOT
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_GOT
BFD_RELOC_TILEGX_IMM16_X0_HW3_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X1_HW3_PLT_PCREL
BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_GD
BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_GD
BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_LE
BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_LE
BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_TLS_LE
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_TLS_LE
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_TLS_LE
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_TLS_LE
BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_TLS_GD
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_TLS_GD
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_TLS_GD
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_TLS_GD

BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_IE
 BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_IE
 BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X0_HW2_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X1_HW2_LAST_PLT_PCREL
 BFD_RELOC_TILEGX_IMM16_X0_HWO_LAST_TLS_IE
 BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_TLS_IE
 BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_TLS_IE
 BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_TLS_IE
 BFD_RELOC_TILEGX_TLS_DTPMOD64
 BFD_RELOC_TILEGX_TLS_DTPOFF64
 BFD_RELOC_TILEGX_TLS_TPOFF64
 BFD_RELOC_TILEGX_TLS_DTPMOD32
 BFD_RELOC_TILEGX_TLS_DTPOFF32
 BFD_RELOC_TILEGX_TLS_TPOFF32
 BFD_RELOC_TILEGX_TLS_GD_CALL
 BFD_RELOC_TILEGX_IMM8_X0_TLS_GD_ADD
 BFD_RELOC_TILEGX_IMM8_X1_TLS_GD_ADD
 BFD_RELOC_TILEGX_IMM8_Y0_TLS_GD_ADD
 BFD_RELOC_TILEGX_IMM8_Y1_TLS_GD_ADD
 BFD_RELOC_TILEGX_TLS_IE_LOAD
 BFD_RELOC_TILEGX_IMM8_X0_TLS_ADD
 BFD_RELOC_TILEGX_IMM8_X1_TLS_ADD
 BFD_RELOC_TILEGX_IMM8_Y0_TLS_ADD
 BFD_RELOC_TILEGX_IMM8_Y1_TLS_ADD

Tilera TILE-Gx Relocations.

BFD_RELOC_BPF_64
 BFD_RELOC_BPF_DISP32
 Linux eBPF relocations.

BFD_RELOC_EPIPHANY_SIMM8
 Adapteva EPIPHANY - 8 bit signed pc-relative displacement

BFD_RELOC_EPIPHANY_SIMM24
 Adapteva EPIPHANY - 24 bit signed pc-relative displacement

BFD_RELOC_EPIPHANY_HIGH
 Adapteva EPIPHANY - 16 most-significant bits of absolute address

BFD_RELOC_EPIPHANY_LOW
 Adapteva EPIPHANY - 16 least-significant bits of absolute address

BFD_RELOC_EPIPHANY_SIMM11
 Adapteva EPIPHANY - 11 bit signed number - add/sub immediate

BFD_RELOC_EIPPHANY_IMM11

Adapteva EIPPHANY - 11 bit sign-magnitude number (ld/st displacement)

BFD_RELOC_EIPPHANY_IMM8

Adapteva EIPPHANY - 8 bit immediate for 16 bit mov instruction.

BFD_RELOC_VISIUM_HI16

BFD_RELOC_VISIUM_LO16

BFD_RELOC_VISIUM_IM16

BFD_RELOC_VISIUM_REL16

BFD_RELOC_VISIUM_HI16_PCREL

BFD_RELOC_VISIUM_LO16_PCREL

BFD_RELOC_VISIUM_IM16_PCREL

Visium Relocations.

BFD_RELOC_WASM32_LEB128

BFD_RELOC_WASM32_LEB128_GOT

BFD_RELOC_WASM32_LEB128_GOT_CODE

BFD_RELOC_WASM32_LEB128_PLT

BFD_RELOC_WASM32_PLT_INDEX

BFD_RELOC_WASM32_ABS32_CODE

BFD_RELOC_WASM32_COPY

BFD_RELOC_WASM32_CODE_POINTER

BFD_RELOC_WASM32_INDEX

BFD_RELOC_WASM32_PLT_SIG

WebAssembly relocations.

BFD_RELOC_CKCORE_NONE

BFD_RELOC_CKCORE_ADDR32

BFD_RELOC_CKCORE_PCREL_IMM8BY4

BFD_RELOC_CKCORE_PCREL_IMM11BY2

BFD_RELOC_CKCORE_PCREL_IMM4BY2

BFD_RELOC_CKCORE_PCREL32

BFD_RELOC_CKCORE_PCREL_JSR_IMM11BY2

BFD_RELOC_CKCORE_GNU_VTINHERIT

BFD_RELOC_CKCORE_GNU_VTENTRY

BFD_RELOC_CKCORE_RELATIVE

BFD_RELOC_CKCORE_COPY

BFD_RELOC_CKCORE_GLOB_DAT

BFD_RELOC_CKCORE_JUMP_SLOT

BFD_RELOC_CKCORE_GOTOFF

BFD_RELOC_CKCORE_GOTPC

BFD_RELOC_CKCORE_GOT32

BFD_RELOC_CKCORE_PLT32

BFD_RELOC_CKCORE_ADDRGOT

BFD_RELOC_CKCORE_ADDRPLT

BFD_RELOC_CKCORE_PCREL_IMM26BY2

BFD_RELOC_CKCORE_PCREL_IMM16BY2

BFD_RELOC_CKCORE_PCREL_IMM16BY4
BFD_RELOC_CKCORE_PCREL_IMM10BY2
BFD_RELOC_CKCORE_PCREL_IMM10BY4
BFD_RELOC_CKCORE_ADDR_HI16
BFD_RELOC_CKCORE_ADDR_LO16
BFD_RELOC_CKCORE_GOTPC_HI16
BFD_RELOC_CKCORE_GOTPC_LO16
BFD_RELOC_CKCORE_GOTOFF_HI16
BFD_RELOC_CKCORE_GOTOFF_LO16
BFD_RELOC_CKCORE_GOT12
BFD_RELOC_CKCORE_GOT_HI16
BFD_RELOC_CKCORE_GOT_LO16
BFD_RELOC_CKCORE_PLT12
BFD_RELOC_CKCORE_PLT_HI16
BFD_RELOC_CKCORE_PLT_LO16
BFD_RELOC_CKCORE_ADDRGOT_HI16
BFD_RELOC_CKCORE_ADDRGOT_LO16
BFD_RELOC_CKCORE_ADDRPLT_HI16
BFD_RELOC_CKCORE_ADDRPLT_LO16
BFD_RELOC_CKCORE_PCREL_JSR_IMM26BY2
BFD_RELOC_CKCORE_TOFFSET_LO16
BFD_RELOC_CKCORE_DOFFSET_LO16
BFD_RELOC_CKCORE_PCREL_IMM18BY2
BFD_RELOC_CKCORE_DOFFSET_IMM18
BFD_RELOC_CKCORE_DOFFSET_IMM18BY2
BFD_RELOC_CKCORE_DOFFSET_IMM18BY4
BFD_RELOC_CKCORE_GOTOFF_IMM18
BFD_RELOC_CKCORE_GOT_IMM18BY4
BFD_RELOC_CKCORE_PLT_IMM18BY4
BFD_RELOC_CKCORE_PCREL_IMM7BY4
BFD_RELOC_CKCORE_TLS_LE32
BFD_RELOC_CKCORE_TLS_IE32
BFD_RELOC_CKCORE_TLS_GD32
BFD_RELOC_CKCORE_TLS_LDM32
BFD_RELOC_CKCORE_TLS_LD032
BFD_RELOC_CKCORE_TLS_DTPMOD32
BFD_RELOC_CKCORE_TLS_DTPOFF32
BFD_RELOC_CKCORE_TLS_TPOFF32
BFD_RELOC_CKCORE_PCREL_FLRW_IMM8BY4
BFD_RELOC_CKCORE_NOJSRI
BFD_RELOC_CKCORE_CALLGRAPH
BFD_RELOC_CKCORE_IRELATIVE
BFD_RELOC_CKCORE_PCREL_BLOOP_IMM4BY4
BFD_RELOC_CKCORE_PCREL_BLOOP_IMM12BY4

C-SKY relocations.

BFD_RELOC_S12Z_OPR
S12Z relocations.

BFD_RELOC_LARCH_TLS_DTPMOD32
BFD_RELOC_LARCH_TLS_DTPREL32
BFD_RELOC_LARCH_TLS_DTPMOD64
BFD_RELOC_LARCH_TLS_DTPREL64
BFD_RELOC_LARCH_TLS_TPREL32
BFD_RELOC_LARCH_TLS_TPREL64
BFD_RELOC_LARCH_MARK_LA
BFD_RELOC_LARCH_MARK_PCREL
BFD_RELOC_LARCH_SOP_PUSH_PCREL
BFD_RELOC_LARCH_SOP_PUSH_ABSOLUTE
BFD_RELOC_LARCH_SOP_PUSH_DUP
BFD_RELOC_LARCH_SOP_PUSH_GPREL
BFD_RELOC_LARCH_SOP_PUSH_TLS_TPREL
BFD_RELOC_LARCH_SOP_PUSH_TLS_GOT
BFD_RELOC_LARCH_SOP_PUSH_TLS_GD
BFD_RELOC_LARCH_SOP_PUSH_PLT_PCREL
BFD_RELOC_LARCH_SOP_ASSERT
BFD_RELOC_LARCH_SOP_NOT
BFD_RELOC_LARCH_SOP_SUB
BFD_RELOC_LARCH_SOP_SL
BFD_RELOC_LARCH_SOP_SR
BFD_RELOC_LARCH_SOP_ADD
BFD_RELOC_LARCH_SOP_AND
BFD_RELOC_LARCH_SOP_IF_ELSE
BFD_RELOC_LARCH_SOP_POP_32_S_10_5
BFD_RELOC_LARCH_SOP_POP_32_U_10_12
BFD_RELOC_LARCH_SOP_POP_32_S_10_12
BFD_RELOC_LARCH_SOP_POP_32_S_10_16
BFD_RELOC_LARCH_SOP_POP_32_S_10_16_S2
BFD_RELOC_LARCH_SOP_POP_32_S_5_20
BFD_RELOC_LARCH_SOP_POP_32_S_0_5_10_16_S2
BFD_RELOC_LARCH_SOP_POP_32_S_0_10_10_16_S2
BFD_RELOC_LARCH_SOP_POP_32_U
BFD_RELOC_LARCH_ADD8
BFD_RELOC_LARCH_ADD16
BFD_RELOC_LARCH_ADD24
BFD_RELOC_LARCH_ADD32
BFD_RELOC_LARCH_ADD64
BFD_RELOC_LARCH_SUB8
BFD_RELOC_LARCH_SUB16
BFD_RELOC_LARCH_SUB24
BFD_RELOC_LARCH_SUB32
BFD_RELOC_LARCH_SUB64
BFD_RELOC_LARCH_B16

BFD_RELOC_LARCH_B21
BFD_RELOC_LARCH_B26
BFD_RELOC_LARCH_ABS_HI20
BFD_RELOC_LARCH_ABS_L012
BFD_RELOC_LARCH_ABS64_L020
BFD_RELOC_LARCH_ABS64_HI12
BFD_RELOC_LARCH_PCALA_HI20
BFD_RELOC_LARCH_PCALA_L012
BFD_RELOC_LARCH_PCALA64_L020
BFD_RELOC_LARCH_PCALA64_HI12
BFD_RELOC_LARCH_GOT_PC_HI20
BFD_RELOC_LARCH_GOT_PC_L012
BFD_RELOC_LARCH_GOT64_PC_L020
BFD_RELOC_LARCH_GOT64_PC_HI12
BFD_RELOC_LARCH_GOT_HI20
BFD_RELOC_LARCH_GOT_L012
BFD_RELOC_LARCH_GOT64_L020
BFD_RELOC_LARCH_GOT64_HI12
BFD_RELOC_LARCH_TLS_LE_HI20
BFD_RELOC_LARCH_TLS_LE_L012
BFD_RELOC_LARCH_TLS_LE64_L020
BFD_RELOC_LARCH_TLS_LE64_HI12
BFD_RELOC_LARCH_TLS_IE_PC_HI20
BFD_RELOC_LARCH_TLS_IE_PC_L012
BFD_RELOC_LARCH_TLS_IE64_PC_L020
BFD_RELOC_LARCH_TLS_IE64_PC_HI12
BFD_RELOC_LARCH_TLS_IE_HI20
BFD_RELOC_LARCH_TLS_IE_L012
BFD_RELOC_LARCH_TLS_IE64_L020
BFD_RELOC_LARCH_TLS_IE64_HI12
BFD_RELOC_LARCH_TLS_LD_PC_HI20
BFD_RELOC_LARCH_TLS_LD_HI20
BFD_RELOC_LARCH_TLS_GD_PC_HI20
BFD_RELOC_LARCH_TLS_GD_HI20
BFD_RELOC_LARCH_32_PCREL
BFD_RELOC_LARCH_RELAX
BFD_RELOC_LARCH_DELETE
BFD_RELOC_LARCH_ALIGN
BFD_RELOC_LARCH_PCREL20_S2
BFD_RELOC_LARCH_CFA
BFD_RELOC_LARCH_ADD6
BFD_RELOC_LARCH_SUB6
BFD_RELOC_LARCH_ADD_ULEB128
BFD_RELOC_LARCH_SUB_ULEB128
BFD_RELOC_LARCH_64_PCREL

LARCH relocations.


```
typedef enum bfd_reloc_code_real bfd_reloc_code_real_type;
```

2.9.2.2 bfd_reloc_type_lookup

```
reloc_howto_type *bfd_reloc_type_lookup (bfd *abfd,          [Function]
    bfd_reloc_code_real_type code); reloc_howto_type
    *bfd_reloc_name_lookup (bfd *abfd, const char *reloc_name);
```

Return a pointer to a howto structure which, when invoked, will perform the relocation *code* on data from the architecture noted.

2.9.2.3 bfd_default_reloc_type_lookup

```
reloc_howto_type *bfd_default_reloc_type_lookup (bfd          [Function]
    *abfd, bfd_reloc_code_real_type code);
```

Provides a default relocation lookup routine for any architecture.

2.9.2.4 bfd_get_reloc_code_name

```
const char *bfd_get_reloc_code_name          [Function]
    (bfd_reloc_code_real_type code);
```

Provides a printable name for the supplied relocation code. Useful mainly for printing error messages.

2.9.2.5 bfd_generic_relax_section

```
bool bfd_generic_relax_section (bfd *abfd, asection          [Function]
    *section, struct bfd_link_info *, bool *);
```

Provides default handling for relaxing for back ends which don't do relaxing.

2.9.2.6 bfd_generic_gc_sections

```
bool bfd_generic_gc_sections (bfd *, struct bfd_link_info    [Function]
    *);
```

Provides default handling for relaxing for back ends which don't do section gc – i.e., does nothing.

2.9.2.7 bfd_generic_lookup_section_flags

```
bool bfd_generic_lookup_section_flags (struct                  [Function]
    bfd_link_info *, struct flag_info *, asection *);
```

Provides default handling for section flags lookup – i.e., does nothing. Returns FALSE if the section should be omitted, otherwise TRUE.

2.9.2.8 bfd_generic_merge_sections

```
bool bfd_generic_merge_sections (bfd *, struct                [Function]
    bfd_link_info *);
```

Provides default handling for SEC_MERGE section merging for back ends which don't have SEC_MERGE support – i.e., does nothing.

2.9.2.9 bfd_generic_get_relocated_section_contents

```
bfd_byte *bfd_generic_get_relocated_section_contents (bfd      [Function]
              *abfd, struct bfd_link_info *link_info, struct
              bfd_link_order *link_order, bfd_byte *data, bool
              relocatable, asymbol **symbols);
```

Provides default handling of relocation effort for back ends which can't be bothered to do it efficiently.

2.9.2.10 _bfd_generic_set_reloc

```
void _bfd_generic_set_reloc (bfd *abfd, sec_ptr section,      [Function]
                             arelent **relptr, unsigned int count);
```

Installs a new set of internal relocations in SECTION.

2.9.2.11 _bfd_unrecognized_reloc

```
bool _bfd_unrecognized_reloc (bfd * abfd, sec_ptr section,    [Function]
                              unsigned int r_type);
```

Reports an unrecognized reloc. Written as a function in order to reduce code duplication. Returns FALSE so that it can be called from a return statement.

2.10 Core files

2.10.1 Core file functions

These are functions pertaining to core files.

2.10.1.1 bfd_core_file_failing_command

```
const char *bfd_core_file_failing_command (bfd *abfd);        [Function]
Return a read-only string explaining which program was running when it failed and
produced the core file abfd.
```

2.10.1.2 bfd_core_file_failing_signal

```
int bfd_core_file_failing_signal (bfd *abfd);                 [Function]
Returns the signal number which caused the core dump which generated the file the
BFD abfd is attached to.
```

2.10.1.3 bfd_core_file_pid

```
int bfd_core_file_pid (bfd *abfd);                            [Function]
Returns the PID of the process the core dump the BFD abfd is attached to was
generated from.
```

2.10.1.4 core_file_matches_executable_p

```
bool core_file_matches_executable_p (bfd *core_bfd, bfd      [Function]
                                      *exec_bfd);
```

Return TRUE if the core file attached to *core_bfd* was generated by a run of the executable file attached to *exec_bfd*, FALSE otherwise.

2.10.1.5 generic_core_file_matches_executable_p

`bool generic_core_file_matches_executable_p (bfd [Function]
*core_bfd, bfd *exec_bfd);`

Return TRUE if the core file attached to *core_bfd* was generated by a run of the executable file attached to *exec_bfd*. The match is based on executable basenames only.

Note: When not able to determine the core file failing command or the executable name, we still return TRUE even though we're not sure that core file and executable match. This is to avoid generating a false warning in situations where we really don't know whether they match or not.

2.11 Targets

Each port of BFD to a different machine requires the creation of a target back end. All the back end provides to the root part of BFD is a structure containing pointers to functions which perform certain low level operations on files. BFD translates the applications's requests through a pointer into calls to the back end routines.

When a file is opened with `bfd_openr`, its format and target are unknown. BFD uses various mechanisms to determine how to interpret the file. The operations performed are:

- Create a BFD by calling the internal routine `_bfd_new_bfd`, then call `bfd_find_target` with the target string supplied to `bfd_openr` and the new BFD pointer.
- If a null target string was provided to `bfd_find_target`, look up the environment variable `GNUTARGET` and use that as the target string.
- If the target string is still NULL, or the target string is `default`, then use the first item in the target vector as the target type, and set `target_defaulted` in the BFD to cause `bfd_check_format` to loop through all the targets. See Section 2.11.1 [`bfd_target`], page 138. See Section 2.8 [Formats], page 47.
- Otherwise, inspect the elements in the target vector one by one, until a match on target name is found. When found, use it.
- Otherwise return the error `bfd_error_invalid_target` to `bfd_openr`.
- `bfd_openr` attempts to open the file using `bfd_open_file`, and returns the BFD.

Once the BFD has been opened and the target selected, the file format may be determined. This is done by calling `bfd_check_format` on the BFD with a suggested format. If `target_defaulted` has been set, each possible target type is tried to see if it recognizes the specified format. `bfd_check_format` returns TRUE when the caller guesses right.

2.11.1 bfd_target

This structure contains everything that BFD knows about a target. It includes things like its byte order, name, and which routines to call to do various operations.

Every BFD points to a target structure with its `xvec` member.

The macros below are used to dispatch to functions through the `bfd_target` vector. They are used in a number of macros further down in `bfd.h`, and are also used when calling various routines by hand inside the BFD implementation. The *arglist* argument must be parenthesized; it contains all the arguments to the called function.

They make the documentation (more) unpleasant to read, so if someone wants to fix this and not break the above, please do.

```
#define BFD_SEND(bfd, message, arglist) \
    ((*((bfd)->xvec->message)) arglist)

#ifdef DEBUG_BFD_SEND
#undef BFD_SEND
#define BFD_SEND(bfd, message, arglist) \
    (((bfd) && (bfd)->xvec && (bfd)->xvec->message) ? \
     ((*((bfd)->xvec->message)) arglist) : \
     (bfd_assert (__FILE__, __LINE__), NULL))
#endif
```

For operations which index on the BFD format:

```
#define BFD_SEND_FMT(bfd, message, arglist) \
    (((bfd)->xvec->message[(int) ((bfd)->format)]) arglist)

#ifdef DEBUG_BFD_SEND
#undef BFD_SEND_FMT
#define BFD_SEND_FMT(bfd, message, arglist) \
    (((bfd) && (bfd)->xvec && (bfd)->xvec->message) ? \
     (((bfd)->xvec->message[(int) ((bfd)->format)]) arglist) : \
     (bfd_assert (__FILE__, __LINE__), NULL))
#endif

/* Defined to TRUE if unused section symbol should be kept. */
#ifndef TARGET_KEEP_UNUSED_SECTION_SYMBOLS
#define TARGET_KEEP_UNUSED_SECTION_SYMBOLS true
#endif
```

This is the structure which defines the type of BFD this is. The `xvec` member of the struct `bfd` itself points here. Each module that implements access to a different target under BFD, defines one of these.

FIXME, these names should be rationalised with the names of the entry points which call them. Too bad we can't have one macro to define them both!

```
typedef struct bfd_target
{
    /* Identifies the kind of target, e.g., SunOS4, Ultrix, etc. */
    const char *name;

    /* The "flavour" of a back end is a general indication about
       the contents of a file. */
    enum bfd_flavour flavour;

    /* The order of bytes within the data area of a file. */
    enum bfd_endian byteorder;
```

```

/* The order of bytes within the header parts of a file. */
enum bfd_endian header_byteorder;

/* A mask of all the flags which an executable may have set -
   from the set BFD_NO_FLAGS, HAS_RELOC, ...D_PAGED. */
flagword object_flags;

/* A mask of all the flags which a section may have set - from
   the set SEC_NO_FLAGS, SEC_ALLOC, ...SET_NEVER_LOAD. */
flagword section_flags;

/* The character normally found at the front of a symbol.
   (if any), perhaps '_'. */
char symbol_leading_char;

/* The pad character for file names within an archive header. */
char ar_pad_char;

/* The maximum number of characters in an archive header. */
unsigned char ar_max_namelen;

/* How well this target matches, used to select between various
   possible targets when more than one target matches. */
unsigned char match_priority;

/* TRUE if unused section symbols should be kept. */
bool keep_unused_section_symbols;

/* Entries for byte swapping for data. These are different from the
   other entry points, since they don't take a BFD as the first argument.
   Certain other handlers could do the same. */
uint64_t      (*bfd_getx64) (const void *);
int64_t       (*bfd_getx_signed_64) (const void *);
void          (*bfd_putx64) (uint64_t, void *);
bfd_vma       (*bfd_getx32) (const void *);
bfd_signed_vma (*bfd_getx_signed_32) (const void *);
void          (*bfd_putx32) (bfd_vma, void *);
bfd_vma       (*bfd_getx16) (const void *);
bfd_signed_vma (*bfd_getx_signed_16) (const void *);
void          (*bfd_putx16) (bfd_vma, void *);

/* Byte swapping for the headers. */
uint64_t      (*bfd_h_getx64) (const void *);
int64_t       (*bfd_h_getx_signed_64) (const void *);
void          (*bfd_h_putx64) (uint64_t, void *);
bfd_vma       (*bfd_h_getx32) (const void *);

```

```

bfd_signed_vma (*bfd_h_getx_signed_32) (const void *);
void           (*bfd_h_putx32) (bfd_vma, void *);
bfd_vma       (*bfd_h_getx16) (const void *);
bfd_signed_vma (*bfd_h_getx_signed_16) (const void *);
void          (*bfd_h_putx16) (bfd_vma, void *);

/* Format dependent routines: these are vectors of entry points
   within the target vector structure, one for each format to check. */

/* Check the format of a file being read. Return a bfd_cleanup on
   success or zero on failure. */
bfd_cleanup (*_bfd_check_format[bfd_type_end]) (bfd *);

/* Set the format of a file being written. */
bool (*_bfd_set_format[bfd_type_end]) (bfd *);

/* Write cached information into a file being written, at bfd_close. */
bool (*_bfd_write_contents[bfd_type_end]) (bfd *);

```

The general target vector. These vectors are initialized using the BFD_JUMP_TABLE macros.

```

/* Generic entry points. */
#define BFD_JUMP_TABLE_GENERIC(NAME) \
  NAME##_close_and_cleanup, \
  NAME##_bfd_free_cached_info, \
  NAME##_new_section_hook, \
  NAME##_get_section_contents, \
  NAME##_get_section_contents_in_window

/* Called when the BFD is being closed to do any necessary cleanup. */
bool (*_close_and_cleanup) (bfd *);
/* Ask the BFD to free all cached information. */
bool (*_bfd_free_cached_info) (bfd *);
/* Called when a new section is created. */
bool (*_new_section_hook) (bfd *, sec_ptr);
/* Read the contents of a section. */
bool (*_bfd_get_section_contents) (bfd *, sec_ptr, void *, file_ptr,
                                   bfd_size_type);
bool (*_bfd_get_section_contents_in_window) (bfd *, sec_ptr, bfd_window *,
                                             file_ptr, bfd_size_type);

/* Entry points to copy private data. */
#define BFD_JUMP_TABLE_COPY(NAME) \
  NAME##_bfd_copy_private_bfd_data, \
  NAME##_bfd_merge_private_bfd_data, \
  _bfd_generic_init_private_section_data, \

```

```

NAME##_bfd_copy_private_section_data, \
NAME##_bfd_copy_private_symbol_data, \
NAME##_bfd_copy_private_header_data, \
NAME##_bfd_set_private_flags, \
NAME##_bfd_print_private_bfd_data

/* Called to copy BFD general private data from one object file
   to another. */
bool (*_bfd_copy_private_bfd_data) (bfd *, bfd *);
/* Called to merge BFD general private data from one object file
   to a common output file when linking. */
bool (*_bfd_merge_private_bfd_data) (bfd *, struct bfd_link_info *);
/* Called to initialize BFD private section data from one object file
   to another. */
#define bfd_init_private_section_data(ibfd, isec, obfd, osec, link_info) \
    BFD_SEND (obfd, _bfd_init_private_section_data, \
              (ibfd, isec, obfd, osec, link_info))
bool (*_bfd_init_private_section_data) (bfd *, sec_ptr, bfd *, sec_ptr, \
                                         struct bfd_link_info *);
/* Called to copy BFD private section data from one object file
   to another. */
bool (*_bfd_copy_private_section_data) (bfd *, sec_ptr, bfd *, sec_ptr);
/* Called to copy BFD private symbol data from one symbol
   to another. */
bool (*_bfd_copy_private_symbol_data) (bfd *, asymbol *,
                                         bfd *, asymbol *);
/* Called to copy BFD private header data from one object file
   to another. */
bool (*_bfd_copy_private_header_data) (bfd *, bfd *);
/* Called to set private backend flags. */
bool (*_bfd_set_private_flags) (bfd *, flagword);

/* Called to print private BFD data. */
bool (*_bfd_print_private_bfd_data) (bfd *, void *);

/* Core file entry points. */
#define BFD_JUMP_TABLE_CORE(NAME) \
    NAME##_core_file_failing_command, \
    NAME##_core_file_failing_signal, \
    NAME##_core_file_matches_executable_p, \
    NAME##_core_file_pid

char (*_core_file_failing_command) (bfd *);
int (*_core_file_failing_signal) (bfd *);
bool (*_core_file_matches_executable_p) (bfd *, bfd *);
int (*_core_file_pid) (bfd *);

```

```

/* Archive entry points. */
#define BFD_JUMP_TABLE_ARCHIVE(NAME) \
  NAME##_slurp_armap, \
  NAME##_slurp_extended_name_table, \
  NAME##_construct_extended_name_table, \
  NAME##_truncate_arname, \
  NAME##_write_armap, \
  NAME##_read_ar_hdr, \
  NAME##_write_ar_hdr, \
  NAME##_openr_next_archived_file, \
  NAME##_get_elt_at_index, \
  NAME##_generic_stat_arch_elt, \
  NAME##_update_armap_timestamp

  bool (*_bfd_slurp_armap) (bfd *);
  bool (*_bfd_slurp_extended_name_table) (bfd *);
  bool (*_bfd_construct_extended_name_table) (bfd *, char **,
                                              bfd_size_type *,
                                              const char **);

  void (*_bfd_truncate_arname) (bfd *, const char *, char *);
  bool (*write_armap) (bfd *, unsigned, struct orl *, unsigned, int);
  void (*_bfd_read_ar_hdr_fn) (bfd *);
  bool (*_bfd_write_ar_hdr_fn) (bfd *, bfd *);
  bfd *(*openr_next_archived_file) (bfd *, bfd *);
#define bfd_get_elt_at_index(b,i) \
  BFD_SEND (b, _bfd_get_elt_at_index, (b,i))
  bfd *(*_bfd_get_elt_at_index) (bfd *, symindex);
  int (*_bfd_stat_arch_elt) (bfd *, struct stat *);
  bool (*_bfd_update_armap_timestamp) (bfd *);

/* Entry points used for symbols. */
#define BFD_JUMP_TABLE_SYMBOLS(NAME) \
  NAME##_get_symtab_upper_bound, \
  NAME##_canonicalize_symtab, \
  NAME##_make_empty_symbol, \
  NAME##_print_symbol, \
  NAME##_get_symbol_info, \
  NAME##_get_symbol_version_string, \
  NAME##_bfd_is_local_label_name, \
  NAME##_bfd_is_target_special_symbol, \
  NAME##_get_lineno, \
  NAME##_find_nearest_line, \
  NAME##_find_nearest_line_with_alt, \
  NAME##_find_line, \
  NAME##_find_inliner_info, \
  NAME##_bfd_make_debug_symbol, \
  NAME##_read_minisymbols, \

```



```

NAME##_minisymbol_to_symbol

long (*_bfd_get_symtab_upper_bound) (bfd *);
long (*_bfd_canonicalize_symtab) (bfd *, struct bfd_symbol **);
struct bfd_symbol *
    (*_bfd_make_empty_symbol) (bfd *);
void (*_bfd_print_symbol) (bfd *, void *, struct bfd_symbol *,
    bfd_print_symbol_type);
#define bfd_print_symbol(b,p,s,e) \
    BFD_SEND (b, _bfd_print_symbol, (b,p,s,e))
void (*_bfd_get_symbol_info) (bfd *, struct bfd_symbol *, symbol_info *);
#define bfd_get_symbol_info(b,p,e) \
    BFD_SEND (b, _bfd_get_symbol_info, (b,p,e))
const char *
    (*_bfd_get_symbol_version_string) (bfd *, struct bfd_symbol *,
    bool, bool *);
#define bfd_get_symbol_version_string(b,s,p,h) \
    BFD_SEND (b, _bfd_get_symbol_version_string, (b,s,p,h))
bool (*_bfd_is_local_label_name) (bfd *, const char *);
bool (*_bfd_is_target_special_symbol) (bfd *, asymbol *);
alint *
    (*_get_lineno) (bfd *, struct bfd_symbol *);
bool (*_bfd_find_nearest_line) (bfd *, struct bfd_symbol **,
    struct bfd_section *, bfd_vma,
    const char **, const char **,
    unsigned int *, unsigned int *);
bool (*_bfd_find_nearest_line_with_alt) (bfd *, const char *,
    struct bfd_symbol **,
    struct bfd_section *, bfd_vma,
    const char **, const char **,
    unsigned int *, unsigned int *);
bool (*_bfd_find_line) (bfd *, struct bfd_symbol **,
    struct bfd_symbol *, const char **,
    unsigned int *);
bool (*_bfd_find_inliner_info)
    (bfd *, const char **, const char **, unsigned int *);
/* Back-door to allow format-aware applications to create debug symbols
   while using BFD for everything else.  Currently used by the assembler
   when creating COFF files.  */
asymbol *
    (*_bfd_make_debug_symbol) (bfd *);
#define bfd_read_minisymbols(b, d, m, s) \
    BFD_SEND (b, _read_minisymbols, (b, d, m, s))
long (*_read_minisymbols) (bfd *, bool, void **, unsigned int *);
#define bfd_minisymbol_to_symbol(b, d, m, f) \
    BFD_SEND (b, _minisymbol_to_symbol, (b, d, m, f))
asymbol *

```

```

        (*_minisymbol_to_symbol) (bfd *, bool, const void *, asymbol *);

/* Routines for relocs. */
#define BFD_JUMP_TABLE_RELOCS(NAME) \
    NAME##_get_reloc_upper_bound, \
    NAME##_canonicalize_reloc, \
    NAME##_set_reloc, \
    NAME##_bfd_reloc_type_lookup, \
    NAME##_bfd_reloc_name_lookup

    long (*_get_reloc_upper_bound) (bfd *, sec_ptr);
    long (*_bfd_canonicalize_reloc) (bfd *, sec_ptr, arelent **,
                                     struct bfd_symbol **);
    void (*_bfd_set_reloc) (bfd *, sec_ptr, arelent **, unsigned int);
/* See documentation on reloc types. */
    reloc_howto_type *
        (*_reloc_type_lookup) (bfd *, bfd_reloc_code_real_type);
    reloc_howto_type *
        (*_reloc_name_lookup) (bfd *, const char *);

/* Routines used when writing an object file. */
#define BFD_JUMP_TABLE_WRITE(NAME) \
    NAME##_set_arch_mach, \
    NAME##_set_section_contents

    bool (*_bfd_set_arch_mach) (bfd *, enum bfd_architecture,
                                unsigned long);
    bool (*_bfd_set_section_contents) (bfd *, sec_ptr, const void *,
                                       file_ptr, bfd_size_type);

/* Routines used by the linker. */
#define BFD_JUMP_TABLE_LINK(NAME) \
    NAME##_sizeof_headers, \
    NAME##_bfd_get_relocated_section_contents, \
    NAME##_bfd_relax_section, \
    NAME##_bfd_link_hash_table_create, \
    NAME##_bfd_link_add_symbols, \
    NAME##_bfd_link_just_syms, \
    NAME##_bfd_copy_link_hash_symbol_type, \
    NAME##_bfd_final_link, \
    NAME##_bfd_link_split_section, \
    NAME##_bfd_link_check_relocs, \
    NAME##_bfd_gc_sections, \
    NAME##_bfd_lookup_section_flags, \
    NAME##_bfd_merge_sections, \
    NAME##_bfd_is_group_section, \
    NAME##_bfd_group_name, \

```

```

NAME##_bfd_discard_group, \
NAME##_section_already_linked, \
NAME##_bfd_define_common_symbol, \
NAME##_bfd_link_hide_symbol, \
NAME##_bfd_define_start_stop

int (*_bfd_sizeof_headers) (bfd *, struct bfd_link_info *);
bfd_byte *
    (*_bfd_get_relocated_section_contents) (bfd *,
                                           struct bfd_link_info *,
                                           struct bfd_link_order *,
                                           bfd_byte *, bool,
                                           struct bfd_symbol **);

bool (*_bfd_relax_section) (bfd *, struct bfd_section *,
                           struct bfd_link_info *, bool *);

/* Create a hash table for the linker. Different backends store
   different information in this table. */
struct bfd_link_hash_table *
    (*_bfd_link_hash_table_create) (bfd *);

/* Add symbols from this object file into the hash table. */
bool (*_bfd_link_add_symbols) (bfd *, struct bfd_link_info *);

/* Indicate that we are only retrieving symbol values from this section. */
void (*_bfd_link_just_syms) (asection *, struct bfd_link_info *);

/* Copy the symbol type and other attributes for a linker script
   assignment of one symbol to another. */
#define bfd_copy_link_hash_symbol_type(b, t, f) \
    BFD_SEND (b, _bfd_copy_link_hash_symbol_type, (b, t, f))
void (*_bfd_copy_link_hash_symbol_type) (bfd *,
                                          struct bfd_link_hash_entry *,
                                          struct bfd_link_hash_entry *);

/* Do a link based on the link_order structures attached to each
   section of the BFD. */
bool (*_bfd_final_link) (bfd *, struct bfd_link_info *);

/* Should this section be split up into smaller pieces during linking. */
bool (*_bfd_link_split_section) (bfd *, struct bfd_section *);

/* Check the relocations in the bfd for validity. */
bool (*_bfd_link_check_relocs) (bfd *, struct bfd_link_info *);

/* Remove sections that are not referenced from the output. */

```

```

bool (*_bfd_gc_sections) (bfd *, struct bfd_link_info *);

/* Sets the bitmask of allowed and disallowed section flags. */
bool (*_bfd_lookup_section_flags) (struct bfd_link_info *,
                                   struct flag_info *, asection *);

/* Attempt to merge SEC_MERGE sections. */
bool (*_bfd_merge_sections) (bfd *, struct bfd_link_info *);

/* Is this section a member of a group? */
bool (*_bfd_is_group_section) (bfd *, const struct bfd_section *);

/* The group name, if section is a member of a group. */
const char *(*_bfd_group_name) (bfd *, const struct bfd_section *);

/* Discard members of a group. */
bool (*_bfd_discard_group) (bfd *, struct bfd_section *);

/* Check if SEC has been already linked during a relocatable or
   final link. */
bool (*_section_already_linked) (bfd *, asection *,
                                 struct bfd_link_info *);

/* Define a common symbol. */
bool (*_bfd_define_common_symbol) (bfd *, struct bfd_link_info *,
                                   struct bfd_link_hash_entry *);

/* Hide a symbol. */
void (*_bfd_link_hide_symbol) (bfd *, struct bfd_link_info *,
                               struct bfd_link_hash_entry *);

/* Define a __start, __stop, .startof. or .sizeof. symbol. */
struct bfd_link_hash_entry *
    (*_bfd_define_start_stop) (struct bfd_link_info *, const char *,
                              asection *);

/* Routines to handle dynamic symbols and relocs. */
#define BFD_JUMP_TABLE_DYNAMIC(NAME) \
    NAME##_get_dynamic_syntab_upper_bound, \
    NAME##_canonicalize_dynamic_syntab, \
    NAME##_get_synthetic_syntab, \
    NAME##_get_dynamic_reloc_upper_bound, \
    NAME##_canonicalize_dynamic_reloc

/* Get the amount of memory required to hold the dynamic symbols. */
long (*_bfd_get_dynamic_syntab_upper_bound) (bfd *);
/* Read in the dynamic symbols. */

```

```

long (*_bfd_canonicalize_dynamic_syntab) (bfd *, struct bfd_symbol **);
/* Create synthesized symbols. */
long (*_bfd_get_synthetic_syntab) (bfd *, long, struct bfd_symbol **,
                                   long, struct bfd_symbol **,
                                   struct bfd_symbol **);
/* Get the amount of memory required to hold the dynamic relocs. */
long (*_bfd_get_dynamic_reloc_upper_bound) (bfd *);
/* Read in the dynamic relocs. */
long (*_bfd_canonicalize_dynamic_reloc) (bfd *, arelent **,
                                         struct bfd_symbol **);

```

A pointer to an alternative `bfd_target` in case the current one is not satisfactory. This can happen when the target cpu supports both big and little endian code, and target chosen by the linker has the wrong endianness. The function `open_output()` in `ld/ldlang.c` uses this field to find an alternative output format that is suitable.

```

/* Opposite endian version of this target. */
const struct bfd_target *alternative_target;

/* Data for use by back-end routines, which isn't
   generic enough to belong in this structure. */
const void *backend_data;

} bfd_target;

static inline const char *
bfd_get_target (const bfd *abfd)
{
    return abfd->xvec->name;
}

static inline enum bfd_flavour
bfd_get_flavour (const bfd *abfd)
{
    return abfd->xvec->flavour;
}

static inline flagword
bfd_applicable_file_flags (const bfd *abfd)
{
    return abfd->xvec->object_flags;
}

static inline bool
bfd_family_coff (const bfd *abfd)
{
    return (bfd_get_flavour (abfd) == bfd_target_coff_flavour

```

```
        || bfd_get_flavour (abfd) == bfd_target_xcoff_flavour);
    }

    static inline bool
    bfd_big_endian (const bfd *abfd)
    {
        return abfd->xvec->byteorder == BFD_ENDIAN_BIG;
    }

    static inline bool
    bfd_little_endian (const bfd *abfd)
    {
        return abfd->xvec->byteorder == BFD_ENDIAN_LITTLE;
    }

    static inline bool
    bfd_header_big_endian (const bfd *abfd)
    {
        return abfd->xvec->header_byteorder == BFD_ENDIAN_BIG;
    }

    static inline bool
    bfd_header_little_endian (const bfd *abfd)
    {
        return abfd->xvec->header_byteorder == BFD_ENDIAN_LITTLE;
    }

    static inline flagword
    bfd_applicable_section_flags (const bfd *abfd)
    {
        return abfd->xvec->section_flags;
    }

    static inline char
    bfd_get_symbol_leading_char (const bfd *abfd)
    {
        return abfd->xvec->symbol_leading_char;
    }

    static inline enum bfd_flavour
    bfd_asyymbol_flavour (const asymbol *sy)
    {
        if ((sy->flags & BSF_SYNTHETIC) != 0)
            return bfd_target_unknown_flavour;
        return sy->the_bfd->xvec->flavour;
    }

    static inline bool
```

```

bfd_keep_unused_section_symbols (const bfd *abfd)
{
    return abfd->xvec->keep_unused_section_symbols;
}

```

2.11.1.1 `_bfd_per_xvec_warn`

```

struct per_xvec_message **_bfd_per_xvec_warn (const          [Function]
    bfd_target *, size_t);

```

Return a location for the given target xvec to use for warnings specific to that target. If TARG is NULL, returns the array of per_xvec_message pointers, otherwise if ALLOC is zero, returns a pointer to a pointer to the list of messages for TARG, otherwise (both TARG and ALLOC non-zero), allocates a new per_xvec_message with space for a string of ALLOC bytes and returns a pointer to a pointer to it. May return a pointer to a NULL pointer on allocation failure.

2.11.1.2 `bfd_set_default_target`

```

bool bfd_set_default_target (const char *name);             [Function]

```

Set the default target vector to use when recognizing a BFD. This takes the name of the target, which may be a BFD target name or a configuration triplet.

2.11.1.3 `bfd_find_target`

```

const bfd_target *bfd_find_target (const char              [Function]
    *target_name, bfd *abfd);

```

Return a pointer to the transfer vector for the object target named *target_name*. If *target_name* is NULL, choose the one in the environment variable GNUTARGET; if that is null or not defined, then choose the first entry in the target list. Passing in the string "default" or setting the environment variable to "default" will cause the first entry in the target list to be returned, and "target_defaulted" will be set in the BFD if *abfd* isn't NULL. This causes `bfd_check_format` to loop over all the targets to find the one that matches the file being read.

2.11.1.4 `bfd_get_target_info`

```

const bfd_target *bfd_get_target_info (const char          [Function]
    *target_name, bfd *abfd, bool *is_bigendian, int
    *underscoring, const char **def_target_arch);

```

Return a pointer to the transfer vector for the object target named *target_name*. If *target_name* is NULL, choose the one in the environment variable GNUTARGET; if that is null or not defined, then choose the first entry in the target list. Passing in the string "default" or setting the environment variable to "default" will cause the first entry in the target list to be returned, and "target_defaulted" will be set in the BFD if *abfd* isn't NULL. This causes `bfd_check_format` to loop over all the targets to find the one that matches the file being read. If *is_bigendian* is not NULL, then set this value to target's endian mode. True for big-endian, FALSE for little-endian or for invalid target. If *underscoring* is not NULL, then set this value to target's underscoring

mode. Zero for none-underscoring, -1 for invalid target, else the value of target vector's symbol underscoring. If *def_target_arch* is not NULL, then set it to the architecture string specified by the *target_name*.

2.11.1.5 bfd_target_list

```
const char ** bfd_target_list (void);
```

 [Function]

Return a freshly malloced NULL-terminated vector of the names of all the valid BFD targets. Do not modify the names.

2.11.1.6 bfd_iterate_over_targets

```
const bfd_target *bfd_iterate_over_targets (int (*func)
      (const bfd_target *, void *), void *data);
```

 [Function]

Call *func* for each target in the list of BFD target vectors, passing *data* to *func*. Stop iterating if *func* returns a non-zero result, and return that target vector. Return NULL if *func* always returns zero.

2.11.1.7 bfd_flavour_name

```
const char *bfd_flavour_name (enum bfd_flavour flavour);
```

 [Function]

Return the string form of *flavour*.

2.12 Architectures

BFD keeps one atom in a BFD describing the architecture of the data attached to the BFD: a pointer to a `bfd_arch_info_type`.

Pointers to structures can be requested independently of a BFD so that an architecture's information can be interrogated without access to an open BFD.

The architecture information is provided by each architecture package. The set of default architectures is selected by the macro `SELECT_ARCHITECTURES`. This is normally set up in the `config/target.mt` file of your choice. If the name is not defined, then all the architectures supported are included.

When BFD starts up, all the architectures are called with an initialize method. It is up to the architecture back end to insert as many items into the list of architectures as it wants to; generally this would be one for each machine and one for the default case (an item with a machine field of 0).

BFD's idea of an architecture is implemented in `archures.c`.

2.12.1 bfd_architecture

This enum gives the object file's CPU architecture, in a global sense—i.e., what processor family does it belong to? Another field indicates which processor within the family is in use. The machine gives a number which distinguishes different versions of the architecture, containing, for example, 68020 for Motorola 68020.

```
enum bfd_architecture
{
    bfd_arch_unknown, /* File arch not known. */

```



```

    bfd_arch_obscure, /* Arch known, not one of these. */
    bfd_arch_m68k,    /* Motorola 68xxx. */
#define bfd_mach_m68000      1
#define bfd_mach_m68008      2
#define bfd_mach_m68010      3
#define bfd_mach_m68020      4
#define bfd_mach_m68030      5
#define bfd_mach_m68040      6
#define bfd_mach_m68060      7
#define bfd_mach_cpu32       8
#define bfd_mach_fido        9
#define bfd_mach_mcf_isa_a_nodiv 10
#define bfd_mach_mcf_isa_a    11
#define bfd_mach_mcf_isa_a_mac 12
#define bfd_mach_mcf_isa_a_emac 13
#define bfd_mach_mcf_isa_aplus 14
#define bfd_mach_mcf_isa_aplus_mac 15
#define bfd_mach_mcf_isa_aplus_emac 16
#define bfd_mach_mcf_isa_b_nousp 17
#define bfd_mach_mcf_isa_b_nousp_mac 18
#define bfd_mach_mcf_isa_b_nousp_emac 19
#define bfd_mach_mcf_isa_b    20
#define bfd_mach_mcf_isa_b_mac 21
#define bfd_mach_mcf_isa_b_emac 22
#define bfd_mach_mcf_isa_b_float 23
#define bfd_mach_mcf_isa_b_float_mac 24
#define bfd_mach_mcf_isa_b_float_emac 25
#define bfd_mach_mcf_isa_c    26
#define bfd_mach_mcf_isa_c_mac 27
#define bfd_mach_mcf_isa_c_emac 28
#define bfd_mach_mcf_isa_c_nodiv 29
#define bfd_mach_mcf_isa_c_nodiv_mac 30
#define bfd_mach_mcf_isa_c_nodiv_emac 31
    bfd_arch_vax,      /* DEC Vax. */

    bfd_arch_or1k,    /* OpenRISC 1000. */
#define bfd_mach_or1k      1
#define bfd_mach_or1knd    2

    bfd_arch_sparc,   /* SPARC. */
#define bfd_mach_sparc      1
/* The difference between v8plus and v9 is that v9 is a true 64 bit env. */
#define bfd_mach_sparc_sparclet 2
#define bfd_mach_sparc_sparclite 3
#define bfd_mach_sparc_v8plus 4
#define bfd_mach_sparc_v8plusa 5 /* with ultrasparc add'ns. */
#define bfd_mach_sparc_sparclite_le 6

```

```

#define bfd_mach_sparc_v9                7
#define bfd_mach_sparc_v9a              8 /* with ultrasparc add'ns. */
#define bfd_mach_sparc_v8plusb          9 /* with cheetah add'ns. */
#define bfd_mach_sparc_v9b             10 /* with cheetah add'ns. */
#define bfd_mach_sparc_v8plusc          11 /* with UA2005 and T1 add'ns. */
#define bfd_mach_sparc_v9c             12 /* with UA2005 and T1 add'ns. */
#define bfd_mach_sparc_v8plusd          13 /* with UA2007 and T3 add'ns. */
#define bfd_mach_sparc_v9d             14 /* with UA2007 and T3 add'ns. */
#define bfd_mach_sparc_v8pluse          15 /* with OSA2001 and T4 add'ns (no IMA). */
#define bfd_mach_sparc_v9e             16 /* with OSA2001 and T4 add'ns (no IMA). */
#define bfd_mach_sparc_v8plusv          17 /* with OSA2011 and T4 and IMA and FJMAU add
#define bfd_mach_sparc_v9v             18 /* with OSA2011 and T4 and IMA and FJMAU add
#define bfd_mach_sparc_v8plusm          19 /* with OSA2015 and M7 add'ns. */
#define bfd_mach_sparc_v9m             20 /* with OSA2015 and M7 add'ns. */
#define bfd_mach_sparc_v8plusm8         21 /* with OSA2017 and M8 add'ns. */
#define bfd_mach_sparc_v9m8            22 /* with OSA2017 and M8 add'ns. */
/* Nonzero if MACH has the v9 instruction set. */
#define bfd_mach_sparc_v9_p(mach) \
    ((mach) >= bfd_mach_sparc_v8plus && (mach) <= bfd_mach_sparc_v9m8 \
     && (mach) != bfd_mach_sparc_sparclite_le)
/* Nonzero if MACH is a 64 bit sparc architecture. */
#define bfd_mach_sparc_64bit_p(mach) \
    ((mach) >= bfd_mach_sparc_v9 \
     && (mach) != bfd_mach_sparc_v8plusb \
     && (mach) != bfd_mach_sparc_v8plusc \
     && (mach) != bfd_mach_sparc_v8plusd \
     && (mach) != bfd_mach_sparc_v8pluse \
     && (mach) != bfd_mach_sparc_v8plusv \
     && (mach) != bfd_mach_sparc_v8plusm \
     && (mach) != bfd_mach_sparc_v8plusm8)
bfd_arch_spu, /* PowerPC SPU. */
#define bfd_mach_spu                    256
bfd_arch_mips, /* MIPS Rxxxx. */
#define bfd_mach_mips3000              3000
#define bfd_mach_mips3900              3900
#define bfd_mach_mips4000              4000
#define bfd_mach_mips4010              4010
#define bfd_mach_mips4100              4100
#define bfd_mach_mips4111              4111
#define bfd_mach_mips4120              4120
#define bfd_mach_mips4300              4300
#define bfd_mach_mips4400              4400
#define bfd_mach_mips4600              4600
#define bfd_mach_mips4650              4650
#define bfd_mach_mips5000              5000
#define bfd_mach_mips5400              5400
#define bfd_mach_mips5500              5500

```

```

#define bfd_mach_mips5900          5900
#define bfd_mach_mips6000          6000
#define bfd_mach_mips7000          7000
#define bfd_mach_mips8000          8000
#define bfd_mach_mips9000          9000
#define bfd_mach_mips10000         10000
#define bfd_mach_mips12000         12000
#define bfd_mach_mips14000         14000
#define bfd_mach_mips16000         16000
#define bfd_mach_mips16            16
#define bfd_mach_mips5             5
#define bfd_mach_mips_allegrex     10111431 /* octal 'AL', 31. */
#define bfd_mach_mips_loongson_2e  3001
#define bfd_mach_mips_loongson_2f  3002
#define bfd_mach_mips_gs464        3003
#define bfd_mach_mips_gs464e       3004
#define bfd_mach_mips_gs264e       3005
#define bfd_mach_mips_sb1          12310201 /* octal 'SB', 01. */
#define bfd_mach_mips_octeon        6501
#define bfd_mach_mips_octeonp       6601
#define bfd_mach_mips_octeon2       6502
#define bfd_mach_mips_octeon3       6503
#define bfd_mach_mips_xlr           887682 /* decimal 'XLR'. */
#define bfd_mach_mips_interactiv_mr2 736550 /* decimal 'IA2'. */
#define bfd_mach_mipsisa32          32
#define bfd_mach_mipsisa32r2        33
#define bfd_mach_mipsisa32r3        34
#define bfd_mach_mipsisa32r5        36
#define bfd_mach_mipsisa32r6        37
#define bfd_mach_mipsisa64          64
#define bfd_mach_mipsisa64r2        65
#define bfd_mach_mipsisa64r3        66
#define bfd_mach_mipsisa64r5        68
#define bfd_mach_mipsisa64r6        69
#define bfd_mach_mips_micromips     96
    bfd_arch_i386, /* Intel 386. */
#define bfd_mach_i386_intel_syntax   (1 << 0)
#define bfd_mach_i386_i8086         (1 << 1)
#define bfd_mach_i386_i386         (1 << 2)
#define bfd_mach_x86_64            (1 << 3)
#define bfd_mach_x64_32            (1 << 4)
#define bfd_mach_i386_i386_intel_syntax (bfd_mach_i386_i386 | bfd_mach_i386_intel_synt
#define bfd_mach_x86_64_intel_syntax (bfd_mach_x86_64 | bfd_mach_i386_intel_syntax)
#define bfd_mach_x64_32_intel_syntax (bfd_mach_x64_32 | bfd_mach_i386_intel_syntax)
    bfd_arch_iamcu, /* Intel MCU. */
#define bfd_mach_iamcu              (1 << 8)
#define bfd_mach_i386_iamcu        (bfd_mach_i386_i386 | bfd_mach_iamcu)

```

```

#define bfd_mach_i386_iamcu_intel_syntax (bfd_mach_i386_iamcu | bfd_mach_i386_intel_sy
    bfd_arch_romp,      /* IBM ROMP PC/RT.  */
    bfd_arch_convex,   /* Convex.  */
    bfd_arch_m98k,     /* Motorola 98xxx.  */
    bfd_arch_pyramid,  /* Pyramid Technology.  */
    bfd_arch_h8300,    /* Renesas H8/300 (formerly Hitachi H8/300).  */
#define bfd_mach_h8300      1
#define bfd_mach_h8300h    2
#define bfd_mach_h8300s    3
#define bfd_mach_h8300hn   4
#define bfd_mach_h8300sn   5
#define bfd_mach_h8300sx   6
#define bfd_mach_h8300sxn  7
    bfd_arch_pdp11,     /* DEC PDP-11.  */
    bfd_arch_powerpc,   /* PowerPC.  */
#define bfd_mach_ppc      32
#define bfd_mach_ppc64    64
#define bfd_mach_ppc_403   403
#define bfd_mach_ppc_403gc 4030
#define bfd_mach_ppc_405   405
#define bfd_mach_ppc_505   505
#define bfd_mach_ppc_601   601
#define bfd_mach_ppc_602   602
#define bfd_mach_ppc_603   603
#define bfd_mach_ppc_ec603e 6031
#define bfd_mach_ppc_604   604
#define bfd_mach_ppc_620   620
#define bfd_mach_ppc_630   630
#define bfd_mach_ppc_750   750
#define bfd_mach_ppc_860   860
#define bfd_mach_ppc_a35   35
#define bfd_mach_ppc_rs64ii 642
#define bfd_mach_ppc_rs64iii 643
#define bfd_mach_ppc_7400  7400
#define bfd_mach_ppc_e500  500
#define bfd_mach_ppc_e500mc 5001
#define bfd_mach_ppc_e500mc64 5005
#define bfd_mach_ppc_e550  5006
#define bfd_mach_ppc_e650  5007
#define bfd_mach_ppc_titan 83
#define bfd_mach_ppc_vle   84
    bfd_arch_rs6000,    /* IBM RS/6000.  */
#define bfd_mach_rs6k      6000
#define bfd_mach_rs6k_rs1  6001
#define bfd_mach_rs6k_rsc  6003
#define bfd_mach_rs6k_rs2  6002
    bfd_arch_hppa,      /* HP PA RISC.  */

```

```

#define bfd_mach_hppa10      10
#define bfd_mach_hppa11      11
#define bfd_mach_hppa20      20
#define bfd_mach_hppa20w     25
    bfd_arch_d10v,          /* Mitsubishi D10V. */
#define bfd_mach_d10v      1
#define bfd_mach_d10v_ts2   2
#define bfd_mach_d10v_ts3   3
    bfd_arch_d30v,          /* Mitsubishi D30V. */
    bfd_arch_dlx,           /* DLX. */
    bfd_arch_m68hc11,       /* Motorola 68HC11. */
    bfd_arch_m68hc12,       /* Motorola 68HC12. */
#define bfd_mach_m6812_default 0
#define bfd_mach_m6812      1
#define bfd_mach_m6812s     2
    bfd_arch_m9s12x,        /* Freescale S12X. */
    bfd_arch_m9s12xg,       /* Freescale XGATE. */
    bfd_arch_s12z,         /* Freescale S12Z. */
#define bfd_mach_s12z_default 0
    bfd_arch_z8k,           /* Zilog Z8000. */
#define bfd_mach_z8001      1
#define bfd_mach_z8002      2
    bfd_arch_sh,           /* Renesas / SuperH SH (formerly Hitachi SH). */
#define bfd_mach_sh          1
#define bfd_mach_sh2        0x20
#define bfd_mach_sh_dsp     0x2d
#define bfd_mach_sh2a       0x2a
#define bfd_mach_sh2a_nofpu  0x2b
#define bfd_mach_sh2a_nofpu_or_sh4_nommu_nofpu 0x2a1
#define bfd_mach_sh2a_nofpu_or_sh3_nommu     0x2a2
#define bfd_mach_sh2a_or_sh4                 0x2a3
#define bfd_mach_sh2a_or_sh3e                0x2a4
#define bfd_mach_sh2e                        0x2e
#define bfd_mach_sh3                         0x30
#define bfd_mach_sh3_nommu                   0x31
#define bfd_mach_sh3_dsp                     0x3d
#define bfd_mach_sh3e                        0x3e
#define bfd_mach_sh4                         0x40
#define bfd_mach_sh4_nofpu                   0x41
#define bfd_mach_sh4_nommu_nofpu            0x42
#define bfd_mach_sh4a                       0x4a
#define bfd_mach_sh4a_nofpu                 0x4b
#define bfd_mach_sh4al_dsp                  0x4d
    bfd_arch_alpha,          /* Dec Alpha. */
#define bfd_mach_alpha_ev4    0x10
#define bfd_mach_alpha_ev5    0x20
#define bfd_mach_alpha_ev6    0x30

```

```

    bfd_arch_arm,      /* Advanced Risc Machines ARM.  */
#define bfd_mach_arm_unknown  0
#define bfd_mach_arm_2        1
#define bfd_mach_arm_2a       2
#define bfd_mach_arm_3        3
#define bfd_mach_arm_3M       4
#define bfd_mach_arm_4        5
#define bfd_mach_arm_4T       6
#define bfd_mach_arm_5        7
#define bfd_mach_arm_5T       8
#define bfd_mach_arm_5TE      9
#define bfd_mach_arm_XScale   10
#define bfd_mach_arm_ep9312   11
#define bfd_mach_arm_iWMMXt   12
#define bfd_mach_arm_iWMMXt2  13
#define bfd_mach_arm_5TEJ     14
#define bfd_mach_arm_6        15
#define bfd_mach_arm_6KZ      16
#define bfd_mach_arm_6T2     17
#define bfd_mach_arm_6K       18
#define bfd_mach_arm_7        19
#define bfd_mach_arm_6M       20
#define bfd_mach_arm_6SM      21
#define bfd_mach_arm_7EM      22
#define bfd_mach_arm_8        23
#define bfd_mach_arm_8R       24
#define bfd_mach_arm_8M_BASE  25
#define bfd_mach_arm_8M_MAIN  26
#define bfd_mach_arm_8_1M_MAIN 27
#define bfd_mach_arm_9        28
    bfd_arch_nds32,    /* Andes NDS32.  */
#define bfd_mach_n1        1
#define bfd_mach_n1h       2
#define bfd_mach_n1h_v2    3
#define bfd_mach_n1h_v3    4
#define bfd_mach_n1h_v3m   5
    bfd_arch_ns32k,    /* National Semiconductors ns32000.  */
    bfd_arch_tic30,    /* Texas Instruments TMS320C30.  */
    bfd_arch_tic4x,    /* Texas Instruments TMS320C3X/4X.  */
#define bfd_mach_tic3x     30
#define bfd_mach_tic4x     40
    bfd_arch_tic54x,    /* Texas Instruments TMS320C54X.  */
    bfd_arch_tic6x,    /* Texas Instruments TMS320C6X.  */
    bfd_arch_v850,      /* NEC V850.  */
    bfd_arch_v850_rh850, /* NEC V850 (using RH850 ABI).  */
#define bfd_mach_v850      1
#define bfd_mach_v850e    'E'

```

```

#define bfd_mach_v850e1      '1'
#define bfd_mach_v850e2      0x4532
#define bfd_mach_v850e2v3    0x45325633
#define bfd_mach_v850e3v5    0x45335635 /* ('E'|'3'|'V'|'5'). */
    bfd_arch_arc,          /* ARC Cores. */
#define bfd_mach_arc_a4      0
#define bfd_mach_arc_a5      1
#define bfd_mach_arc_arc600  2
#define bfd_mach_arc_arc601  4
#define bfd_mach_arc_arc700  3
#define bfd_mach_arc_arcv2   5
    bfd_arch_m32c,        /* Renesas M16C/M32C. */
#define bfd_mach_m16c        0x75
#define bfd_mach_m32c        0x78
    bfd_arch_m32r,        /* Renesas M32R (formerly Mitsubishi M32R/D). */
#define bfd_mach_m32r        1 /* For backwards compatibility. */
#define bfd_mach_m32rx       'x'
#define bfd_mach_m32r2       '2'
    bfd_arch_mn10200,     /* Matsushita MN10200. */
    bfd_arch_mn10300,     /* Matsushita MN10300. */
#define bfd_mach_mn10300     300
#define bfd_mach_am33        330
#define bfd_mach_am33_2     332
    bfd_arch_fr30,
#define bfd_mach_fr30        0x46523330
    bfd_arch_frv,
#define bfd_mach_frv         1
#define bfd_mach_frvsimple    2
#define bfd_mach_fr300       300
#define bfd_mach_fr400       400
#define bfd_mach_fr450       450
#define bfd_mach_frvtomcat   499 /* fr500 prototype. */
#define bfd_mach_fr500       500
#define bfd_mach_fr550       550
    bfd_arch_moxie,      /* The moxie processor. */
#define bfd_mach_moxie       1
    bfd_arch_ft32,      /* The ft32 processor. */
#define bfd_mach_ft32        1
#define bfd_mach_ft32b       2
    bfd_arch_mcore,
    bfd_arch_mep,
#define bfd_mach_mep         1
#define bfd_mach_mep_h1      0x6831
#define bfd_mach_mep_c5      0x6335
    bfd_arch_metag,
#define bfd_mach_metag       1
    bfd_arch_ia64,      /* HP/Intel ia64. */

```

```

#define bfd_mach_ia64_elf64    64
#define bfd_mach_ia64_elf32    32
    bfd_arch_ip2k,          /* Ubicom IP2K microcontrollers. */
#define bfd_mach_ip2022        1
#define bfd_mach_ip2022ext     2
    bfd_arch_iq2000,       /* Vitesse IQ2000. */
#define bfd_mach_iq2000        1
#define bfd_mach_iq10          2
    bfd_arch_bpf,          /* Linux eBPF. */
#define bfd_mach_bpf           1
#define bfd_mach_xbpf          2
    bfd_arch_epiphany,     /* Adapteva EPIPHANY. */
#define bfd_mach_epiphany16    1
#define bfd_mach_epiphany32    2
    bfd_arch_mt,
#define bfd_mach_ms1           1
#define bfd_mach_mrisc2        2
#define bfd_mach_ms2           3
    bfd_arch_pj,
    bfd_arch_avr,          /* Atmel AVR microcontrollers. */
#define bfd_mach_avr1          1
#define bfd_mach_avr2          2
#define bfd_mach_avr25         25
#define bfd_mach_avr3          3
#define bfd_mach_avr31         31
#define bfd_mach_avr35         35
#define bfd_mach_avr4          4
#define bfd_mach_avr5          5
#define bfd_mach_avr51         51
#define bfd_mach_avr6          6
#define bfd_mach_avrtiny       100
#define bfd_mach_avrxmega1     101
#define bfd_mach_avrxmega2     102
#define bfd_mach_avrxmega3     103
#define bfd_mach_avrxmega4     104
#define bfd_mach_avrxmega5     105
#define bfd_mach_avrxmega6     106
#define bfd_mach_avrxmega7     107
    bfd_arch_bfin,         /* ADI Blackfin. */
#define bfd_mach_bfin          1
    bfd_arch_cr16,         /* National Semiconductor CompactRISC (ie CR16). */
#define bfd_mach_cr16          1
    bfd_arch_crx,          /* National Semiconductor CRX. */
#define bfd_mach_crx           1
    bfd_arch_cris,         /* Axis CRIS. */
#define bfd_mach_cris_v0_v10    255
#define bfd_mach_cris_v32      32

```



```
#define bfd_mach_cris_v10_v32 1032
  bfd_arch_riscv,
#define bfd_mach_riscv32      132
#define bfd_mach_riscv64     164
  bfd_arch_rl78,
#define bfd_mach_rl78        0x75
  bfd_arch_rx, /* Renesas RX. */
#define bfd_mach_rx          0x75
#define bfd_mach_rx_v2      0x76
#define bfd_mach_rx_v3      0x77
  bfd_arch_s390, /* IBM s390. */
#define bfd_mach_s390_31     31
#define bfd_mach_s390_64     64
  bfd_arch_score, /* Sunplus score. */
#define bfd_mach_score3      3
#define bfd_mach_score7      7
  bfd_arch_mmix, /* Donald Knuth's educational processor. */
  bfd_arch_xstormy16,
#define bfd_mach_xstormy16   1
  bfd_arch_msp430, /* Texas Instruments MSP430 architecture. */
#define bfd_mach_msp11       11
#define bfd_mach_msp110     110
#define bfd_mach_msp12       12
#define bfd_mach_msp13       13
#define bfd_mach_msp14       14
#define bfd_mach_msp15       15
#define bfd_mach_msp16       16
#define bfd_mach_msp20       20
#define bfd_mach_msp21       21
#define bfd_mach_msp22       22
#define bfd_mach_msp23       23
#define bfd_mach_msp24       24
#define bfd_mach_msp26       26
#define bfd_mach_msp31       31
#define bfd_mach_msp32       32
#define bfd_mach_msp33       33
#define bfd_mach_msp41       41
#define bfd_mach_msp42       42
#define bfd_mach_msp43       43
#define bfd_mach_msp44       44
#define bfd_mach_msp430x     45
#define bfd_mach_msp46       46
#define bfd_mach_msp47       47
#define bfd_mach_msp54       54
  bfd_arch_xgate, /* Freescale XGATE. */
#define bfd_mach_xgate       1
  bfd_arch_xtensa, /* Tensilica's Xtensa cores. */
```

```

#define bfd_mach_xtensa      1
    bfd_arch_z80,
/* Zilog Z80 without undocumented opcodes. */
#define bfd_mach_z80strict  1
/* Zilog Z180: successor with additional instructions, but without
    halves of ix and iy. */
#define bfd_mach_z180      2
/* Zilog Z80 with ixl, ixh, iyl, and iyh. */
#define bfd_mach_z80      3
/* Zilog eZ80 (successor of Z80 & Z180) in Z80 (16-bit address) mode. */
#define bfd_mach_ez80_z80  4
/* Zilog eZ80 (successor of Z80 & Z180) in ADL (24-bit address) mode. */
#define bfd_mach_ez80_adl  5
/* Z8ON */
#define bfd_mach_z80n      6
/* Zilog Z80 with all undocumented instructions. */
#define bfd_mach_z80full   7
/* GameBoy Z80 (reduced instruction set). */
#define bfd_mach_gbz80     8
/* ASCII R800: successor with multiplication. */
#define bfd_mach_r800     11
    bfd_arch_lm32, /* Lattice Mico32. */
#define bfd_mach_lm32     1
    bfd_arch_microblaze, /* Xilinx MicroBlaze. */
    bfd_arch_tilepro, /* Tilera TILEPro. */
    bfd_arch_tilegx, /* Tilera TILE-Gx. */
#define bfd_mach_tilepro  1
#define bfd_mach_tilegx   1
#define bfd_mach_tilegx32 2
    bfd_arch_aarch64, /* AArch64. */
#define bfd_mach_aarch64  0
#define bfd_mach_aarch64_8R 1
#define bfd_mach_aarch64_ilp32 32
#define bfd_mach_aarch64_llp64 64
    bfd_arch_nios2, /* Nios II. */
#define bfd_mach_nios2    0
#define bfd_mach_nios2r1  1
#define bfd_mach_nios2r2  2
    bfd_arch_visium, /* Visium. */
#define bfd_mach_visium   1
    bfd_arch_wasm32, /* WebAssembly. */
#define bfd_mach_wasm32   1
    bfd_arch_pru, /* PRU. */
#define bfd_mach_pru      0
    bfd_arch_nfp, /* Netronome Flow Processor */
#define bfd_mach_nfp3200  0x3200
#define bfd_mach_nfp6000  0x6000

```

```

    bfd_arch_csky,      /* C-SKY.  */
#define bfd_mach_ck_unknown  0
#define bfd_mach_ck510      1
#define bfd_mach_ck610      2
#define bfd_mach_ck801      3
#define bfd_mach_ck802      4
#define bfd_mach_ck803      5
#define bfd_mach_ck807      6
#define bfd_mach_ck810      7
#define bfd_mach_ck860      8
    bfd_arch_loongarch, /* LoongArch */
#define bfd_mach_loongarch32  1
#define bfd_mach_loongarch64  2
    bfd_arch_amdgc,     /* AMDGCN */
#define bfd_mach_amdgcn_unknown 0x000
#define bfd_mach_amdgcn_gfx900  0x02c
#define bfd_mach_amdgcn_gfx904  0x02e
#define bfd_mach_amdgcn_gfx906  0x02f
#define bfd_mach_amdgcn_gfx908  0x030
#define bfd_mach_amdgcn_gfx90a  0x03f
#define bfd_mach_amdgcn_gfx1010 0x033
#define bfd_mach_amdgcn_gfx1011 0x034
#define bfd_mach_amdgcn_gfx1012 0x035
#define bfd_mach_amdgcn_gfx1030 0x036
#define bfd_mach_amdgcn_gfx1031 0x037
#define bfd_mach_amdgcn_gfx1032 0x038
    bfd_arch_last
};

```

2.12.2 bfd_arch_info

This structure contains information on architectures for use within BFD.

```

typedef struct bfd_arch_info
{
    int bits_per_word;
    int bits_per_address;
    int bits_per_byte;
    enum bfd_architecture arch;
    unsigned long mach;
    const char *arch_name;
    const char *printable_name;
    unsigned int section_align_power;
    /* TRUE if this is the default machine for the architecture.
       The default arch should be the first entry for an arch so that
       all the entries for that arch can be accessed via next.  */
    bool the_default;
};

```

```

const struct bfd_arch_info * (*compatible) (const struct bfd_arch_info *,
                                             const struct bfd_arch_info *);

bool (*scan) (const struct bfd_arch_info *, const char *);

/* Allocate via bfd_malloc and return a fill buffer of size COUNT.  If
   IS_BIGENDIAN is TRUE, the order of bytes is big endian.  If CODE is
   TRUE, the buffer contains code.  */
void *(*fill) (bfd_size_type count, bool is_bigendian, bool code);

const struct bfd_arch_info *next;

/* On some architectures the offset for a relocation can point into
   the middle of an instruction.  This field specifies the maximum
   offset such a relocation can have (in octets).  This affects the
   behaviour of the disassembler, since a value greater than zero
   means that it may need to disassemble an instruction twice, once
   to get its length and then a second time to display it.  If the
   value is negative then this has to be done for every single
   instruction, regardless of the offset of the reloc.  */
signed int max_reloc_offset_into_insn;
}
bfd_arch_info_type;

```

2.12.2.1 bfd_printable_name

```
const char *bfd_printable_name (bfd *abfd);
```

[Function]
Return a printable string representing the architecture and machine from the pointer to the architecture info structure.

2.12.2.2 bfd_scan_arch

```
const bfd_arch_info_type *bfd_scan_arch (const char
                                         *string);
```

[Function]
Figure out if BFD supports any cpu which could be described with the name *string*. Return a pointer to an `arch_info` structure if a machine is found, otherwise NULL.

2.12.2.3 bfd_arch_list

```
const char **bfd_arch_list (void);
```

[Function]
Return a freshly malloced NULL-terminated vector of the names of all the valid BFD architectures. Do not modify the names.

2.12.2.4 bfd_arch_get_compatible

```
const bfd_arch_info_type *bfd_arch_get_compatible (const      [Function]
          bfd *abfd, const bfd *bbfd, bool accept_unknowns);
```

Determine whether two BFDs' architectures and machine types are compatible. Calculates the lowest common denominator between the two architectures and machine types implied by the BFDs and returns a pointer to an `arch_info` structure describing the compatible machine.

2.12.2.5 bfd_default_arch_struct

The `bfd_default_arch_struct` is an item of `bfd_arch_info_type` which has been initialized to a fairly generic state. A BFD starts life by pointing to this structure, until the correct back end has determined the real architecture of the file.

```
extern const bfd_arch_info_type bfd_default_arch_struct;
```

2.12.2.6 bfd_set_arch_info

```
void bfd_set_arch_info (bfd *abfd, const                      [Function]
          bfd_arch_info_type *arg);
```

Set the architecture info of *abfd* to *arg*.

2.12.2.7 bfd_default_set_arch_mach

```
bool bfd_default_set_arch_mach (bfd *abfd, enum              [Function]
          bfd_architecture arch, unsigned long mach);
```

Set the architecture and machine type in BFD *abfd* to *arch* and *mach*. Find the correct pointer to a structure and insert it into the `arch_info` pointer.

2.12.2.8 bfd_get_arch

```
enum bfd_architecture bfd_get_arch (const bfd *abfd);        [Function]
```

Return the enumerated type which describes the BFD *abfd*'s architecture.

2.12.2.9 bfd_get_mach

```
unsigned long bfd_get_mach (const bfd *abfd);                [Function]
```

Return the long type which describes the BFD *abfd*'s machine.

2.12.2.10 bfd_arch_bits_per_byte

```
unsigned int bfd_arch_bits_per_byte (const bfd *abfd);       [Function]
```

Return the number of bits in one of the BFD *abfd*'s architecture's bytes.

2.12.2.11 bfd_arch_bits_per_address

```
unsigned int bfd_arch_bits_per_address (const bfd *abfd);    [Function]
```

Return the number of bits in one of the BFD *abfd*'s architecture's addresses.

2.12.2.12 `bfd_default_compatible`

```
const bfd_arch_info_type *bfd_default_compatible (const      [Function]
          bfd_arch_info_type *a, const bfd_arch_info_type *b);
```

The default function for testing for compatibility.

2.12.2.13 `bfd_default_scan`

```
bool bfd_default_scan (const struct bfd_arch_info *info,      [Function]
                      const char *string);
```

The default function for working out whether this is an architecture hit and a machine hit.

2.12.2.14 `bfd_get_arch_info`

```
const bfd_arch_info_type *bfd_get_arch_info (bfd *abfd);      [Function]
```

Return the architecture info struct in *abfd*.

2.12.2.15 `bfd_lookup_arch`

```
const bfd_arch_info_type *bfd_lookup_arch (enum              [Function]
          bfd_architecture arch, unsigned long machine);
```

Look for the architecture info structure which matches the arguments *arch* and *machine*. A machine of 0 matches the machine/architecture structure which marks itself as the default.

2.12.2.16 `bfd_printable_arch_mach`

```
const char *bfd_printable_arch_mach (enum bfd_architecture  [Function]
          arch, unsigned long machine);
```

Return a printable string representing the architecture and machine type.

This routine is deprecated.

2.12.2.17 `bfd_octets_per_byte`

```
unsigned int bfd_octets_per_byte (const bfd *abfd, const      [Function]
          asection *sec);
```

Return the number of octets (8-bit quantities) per target byte (minimum addressable unit). In most cases, this will be one, but some DSP targets have 16, 32, or even 48 bits per byte.

2.12.2.18 `bfd_arch_mach_octets_per_byte`

```
unsigned int bfd_arch_mach_octets_per_byte (enum             [Function]
          bfd_architecture arch, unsigned long machine);
```

See `bfd_octets_per_byte`.

This routine is provided for those cases where a `bfd *` is not available

2.12.2.19 bfd_arch_default_fill

```
void *bfd_arch_default_fill (bfd_size_type count, bool is_bigendian, bool code);
```

[Function]

Allocate via `bfd_malloc` and return a fill buffer of size `COUNT`. If `IS_BIGENDIAN` is `TRUE`, the order of bytes is big endian. If `CODE` is `TRUE`, the buffer contains code.

2.13 Opening and closing BFDs

2.13.1 Functions for opening and closing

2.13.1.1 _bfd_new_bfd

```
bfd *_bfd_new_bfd (void);
```

[Function]

Return a new BFD. All BFD's are allocated through this routine.

2.13.1.2 _bfd_new_bfd_contained_in

```
bfd *_bfd_new_bfd_contained_in (bfd *);
```

[Function]

Allocate a new BFD as a member of archive OBFD.

2.13.1.3 _bfd_free_cached_info

```
bool _bfd_free_cached_info (bfd *);
```

[Function]

Free objalloc memory.

2.13.1.4 bfd_fopen

```
bfd *bfd_fopen (const char *filename, const char *target, const char *mode, int fd);
```

[Function]

Open the file *filename* with the target *target*. Return a pointer to the created BFD. If *fd* is not -1, then `fdopen` is used to open the file; otherwise, `fopen` is used. *mode* is passed directly to `fopen` or `fdopen`.

Calls `bfd_find_target`, so *target* is interpreted as by that function.

The new BFD is marked as cacheable iff *fd* is -1.

If `NULL` is returned then an error has occurred. Possible errors are `bfd_error_no_memory`, `bfd_error_invalid_target` or `system_call` error.

On error, *fd* is always closed.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the `bfd_get_filename()` macro.

2.13.1.5 bfd_openr

```
bfd *bfd_openr (const char *filename, const char *target);
```

[Function]

Open the file *filename* (using `fopen`) with the target *target*. Return a pointer to the created BFD.

Calls `bfd_find_target`, so *target* is interpreted as by that function.

If `NULL` is returned then an error has occurred. Possible errors are `bfd_error_no_memory`, `bfd_error_invalid_target` or `system_call` error.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the `bfd_get_filename()` macro.

2.13.1.6 `bfd_fdopenr`

```
bfd *bfd_fdopenr (const char *filename, const char *target, int fd);
```

[Function]

`bfd_fdopenr` is to `bfd_fopenr` much like `fdopen` is to `fopen`. It opens a BFD on a file already described by the *fd* supplied.

When the file is later `bfd_closed`, the file descriptor will be closed. If the caller desires that this file descriptor be cached by BFD (opened as needed, closed as needed to free descriptors for other opens), with the supplied *fd* used as an initial file descriptor (but subject to closure at any time), call `bfd_set_cacheable(bfd, 1)` on the returned BFD. The default is to assume no caching; the file descriptor will remain open until `bfd_close`, and will not be affected by BFD operations on other files.

Possible errors are `bfd_error_no_memory`, `bfd_error_invalid_target` and `bfd_error_system_call`.

On error, *fd* is closed.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the `bfd_get_filename()` macro.

2.13.1.7 `bfd_fdopenw`

```
bfd *bfd_fdopenw (const char *filename, const char *target, int fd);
```

[Function]

`bfd_fdopenw` is exactly like `bfd_fdopenr` with the exception that the resulting BFD is suitable for output.

2.13.1.8 `bfd_openstreamr`

```
bfd *bfd_openstreamr (const char * filename, const char * target, void * stream);
```

[Function]

Open a BFD for read access on an existing stdio stream. When the BFD is passed to `bfd_close`, the stream will be closed.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the `bfd_get_filename()` macro.

2.13.1.9 bfd_openr_iovec

```
bfd *bfd_openr_iovec (const char *filename, const char          [Function]
                    *target, void *(*open_func) (struct bfd *nbfd, void
                    *open_closure), void *open_closure, file_ptr (*pread_func)
                    (struct bfd *nbfd, void *stream, void *buf, file_ptr nbytes,
                    file_ptr offset), int (*close_func) (struct bfd *nbfd, void
                    *stream), int (*stat_func) (struct bfd *abfd, void *stream,
                    struct stat *sb));
```

Create and return a BFD backed by a read-only *stream*. The *stream* is created using *open_func*, accessed using *pread_func* and destroyed using *close_func*.

Calls *bfd_find_target*, so *target* is interpreted as by that function.

Calls *open_func* (which can call *bfd_zalloc* and *bfd_get_filename*) to obtain the read-only stream backing the BFD. *open_func* either succeeds returning the non-NULL *stream*, or fails returning NULL (setting *bfd_error*).

Calls *pread_func* to request *nbytes* of data from *stream* starting at *offset* (e.g., via a call to *bfd_read*). *pread_func* either succeeds returning the number of bytes read (which can be less than *nbytes* when end-of-file), or fails returning -1 (setting *bfd_error*).

Calls *close_func* when the BFD is later closed using *bfd_close*. *close_func* either succeeds returning 0, or fails returning -1 (setting *bfd_error*).

Calls *stat_func* to fill in a stat structure for *bfd_stat*, *bfd_get_size*, and *bfd_get_mtime* calls. *stat_func* returns 0 on success, or returns -1 on failure (setting *bfd_error*).

If *bfd_openr_iovec* returns NULL then an error has occurred. Possible errors are *bfd_error_no_memory*, *bfd_error_invalid_target* and *bfd_error_system_call*.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the *bfd_get_filename()* macro.

2.13.1.10 bfd_openw

```
bfd *bfd_openw (const char *filename, const char *target);    [Function]
```

Create a BFD, associated with file *filename*, using the file format *target*, and return a pointer to it.

Possible errors are *bfd_error_system_call*, *bfd_error_no_memory*, *bfd_error_invalid_target*.

A copy of the *filename* argument is stored in the newly created BFD. It can be accessed via the *bfd_get_filename()* macro.

2.13.1.11 bfd_elf_bfd_from_remote_memory

```
bfd *bfd_elf_bfd_from_remote_memory (bfd *templ, bfd_vma      [Function]
                                     ehdr_vma, bfd_size_type size, bfd_vma *loadbasep, int
                                     (*target_read_memory) (bfd_vma vma, bfd_byte *myaddr,
                                     bfd_size_type len));
```

Create a new BFD as if by *bfd_openr*. Rather than opening a file, reconstruct an ELF file by reading the segments out of remote memory based on the ELF file header at *EHDR_VMA* and the ELF program headers it points to. If non-zero, *SIZE* is the

known extent of the object. If not null, *LOADBASEP is filled in with the difference between the VMAs from which the segments were read, and the VMAs the file headers (and hence BFD's idea of each section's VMA) put them at.

The function TARGET_READ_MEMORY is called to copy LEN bytes from the remote memory at target address VMA into the local buffer at MYADDR; it should return zero on success or an errno code on failure. TEMPL must be a BFD for an ELF target with the word size and byte order found in the remote memory.

2.13.1.12 bfd_close

`bool bfd_close (bfd *abfd);` [Function]

Close a BFD. If the BFD was open for writing, then pending operations are completed and the file written out and closed. If the created file is executable, then `chmod` is called to mark it as such.

All memory attached to the BFD is released.

The file descriptor associated with the BFD is closed (even if it was passed in to BFD by `bfd_fdopenr`).

TRUE is returned if all is ok, otherwise FALSE.

2.13.1.13 bfd_close_all_done

`bool bfd_close_all_done (bfd *);` [Function]

Close a BFD. Differs from `bfd_close` since it does not complete any pending operations. This routine would be used if the application had just used BFD for swapping and didn't want to use any of the writing code.

If the created file is executable, then `chmod` is called to mark it as such.

All memory attached to the BFD is released.

TRUE is returned if all is ok, otherwise FALSE.

2.13.1.14 bfd_create

`bfd *bfd_create (const char *filename, bfd *templ);` [Function]

Create a new BFD in the manner of `bfd_openw`, but without opening a file. The new BFD takes the target from the target used by `templ`. The format is always set to `bfd_object`.

A copy of the `filename` argument is stored in the newly created BFD. It can be accessed via the `bfd_get_filename()` macro.

2.13.1.15 bfd_make_writable

`bool bfd_make_writable (bfd *abfd);` [Function]

Takes a BFD as created by `bfd_create` and converts it into one like as returned by `bfd_openw`. It does this by converting the BFD to `BFD_IN_MEMORY`. It's assumed that you will call `bfd_make_readable` on this `bfd` later.

TRUE is returned if all is ok, otherwise FALSE.

2.13.1.16 bfd_make_readable

`bool bfd_make_readable (bfd *abfd);` [Function]
 Takes a BFD as created by `bfd_create` and `bfd_make_writable` and converts it into one like as returned by `bfd_openr`. It does this by writing the contents out to the memory buffer, then reversing the direction.
 TRUE is returned if all is ok, otherwise FALSE.

2.13.1.17 bfd_calc_gnu_debuglink_crc32

`uint32_t bfd_calc_gnu_debuglink_crc32 (uint32_t crc, const bfd_byte *buf, bfd_size_type len);` [Function]
 Computes a CRC value as used in the `.gnu_debuglink` section. Advances the previously computed `crc` value by computing and adding in the `crc32` for `len` bytes of `buf`.
 Return the updated CRC32 value.

2.13.1.18 bfd_get_debug_link_info

`char *bfd_get_debug_link_info (bfd *abfd, uint32_t *crc32_out);` [Function]
 Extracts the filename and CRC32 value for any separate debug information file associated with `abfd`.
 Returns the filename of the associated debug information file, or NULL if there is no such file. If the filename was found then the contents of `crc32_out` are updated to hold the corresponding CRC32 value for the file.
 The returned filename is allocated with `malloc`; freeing it is the responsibility of the caller.

2.13.1.19 bfd_get_alt_debug_link_info

`char *bfd_get_alt_debug_link_info (bfd * abfd, bfd_size_type *buildid_len, bfd_byte **buildid_out);` [Function]
 Fetch the filename and BuildID value for any alternate debuginfo associated with `abfd`. Return NULL if no such info found, otherwise return filename and update `buildid_len` and `buildid_out`. The returned filename and `build_id` are allocated with `malloc`; freeing them is the responsibility of the caller.

2.13.1.20 bfd_follow_gnu_debuglink

`char *bfd_follow_gnu_debuglink (bfd *abfd, const char *dir);` [Function]
 Takes a BFD and searches it for a `.gnu_debuglink` section. If this section is found, it examines the section for the name and checksum of a `'debug'` file containing auxiliary debugging information. It then searches the filesystem for this `.debug` file in some standard locations, including the directory tree rooted at `dir`, and if found returns the full filename.
 If `dir` is NULL, the search will take place starting at the current directory.

Returns NULL on any errors or failure to locate the .debug file, otherwise a pointer to a heap-allocated string containing the filename. The caller is responsible for freeing this string.

2.13.1.21 bfd_follow_gnu_debugaltlink

```
char *bfd_follow_gnu_debugaltlink (bfd *abfd, const char      [Function]
    *dir);
```

Takes a BFD and searches it for a .gnu_debugaltlink section. If this section is found, it examines the section for the name of a file containing auxiliary debugging information. It then searches the filesystem for this file in a set of standard locations, including the directory tree rooted at *dir*, and if found returns the full filename.

If *dir* is NULL, the search will take place starting at the current directory.

Returns NULL on any errors or failure to locate the debug file, otherwise a pointer to a heap-allocated string containing the filename. The caller is responsible for freeing this string.

2.13.1.22 bfd_create_gnu_debuglink_section

```
struct bfd_section *bfd_create_gnu_debuglink_section (bfd      [Function]
    *abfd, const char *filename);
```

Takes a *BFD* and adds a .gnu_debuglink section to it. The section is sized to be big enough to contain a link to the specified *filename*.

A pointer to the new section is returned if all is ok. Otherwise NULL is returned and *bfd_error* is set.

2.13.1.23 bfd_fill_in_gnu_debuglink_section

```
bool bfd_fill_in_gnu_debuglink_section (bfd *abfd, struct      [Function]
    bfd_section *sect, const char *filename);
```

Takes a *BFD* and containing a .gnu_debuglink section *SECT* and fills in the contents of the section to contain a link to the specified *filename*. The filename should be absolute or relative to the current directory.

TRUE is returned if all is ok. Otherwise FALSE is returned and *bfd_error* is set.

2.13.1.24 bfd_follow_build_id_debuglink

```
char *bfd_follow_build_id_debuglink (bfd *abfd, const char    [Function]
    *dir);
```

Takes *abfd* and searches it for a .note.gnu.build-id section. If this section is found, it extracts the value of the NT_GNU_BUILD_ID note, which should be a hexadecimal value *NNNN+NN* (for 32+ hex digits). It then searches the filesystem for a file named *.build-id/NN/NN+NN.debug* in a set of standard locations, including the directory tree rooted at *dir*. The filename of the first matching file to be found is returned. A matching file should contain a .note.gnu.build-id section with the same *NNNN+NN* note as *abfd*, although this check is currently not implemented.

If *dir* is NULL, the search will take place starting at the current directory.

Returns NULL on any errors or failure to locate the debug file, otherwise a pointer to a heap-allocated string containing the filename. The caller is responsible for freeing this string.

2.13.1.25 `bfd_set_filename`

```
const char *bfd_set_filename (bfd *abfd, const char      [Function]
                             *filename);
```

Set the filename of *abfd*, copying the FILENAME parameter to `bfd_alloc`'d memory owned by *abfd*. Returns a pointer the newly allocated name, or NULL if the allocation failed.

2.14 File caching

The file caching mechanism is embedded within BFD and allows the application to open as many BFDs as it wants without regard to the underlying operating system's file descriptor limit (often as low as 20 open files). The module in `cache.c` maintains a least recently used list of `bfd_cache_max_open` files, and exports the name `bfd_cache_lookup`, which runs around and makes sure that the required BFD is open. If not, then it chooses a file to close, closes it and opens the one wanted, returning its file handle.

2.14.1 Caching functions

2.14.1.1 `bfd_cache_init`

```
bool bfd_cache_init (bfd *abfd);                          [Function]
    Add a newly opened BFD to the cache.
```

2.14.1.2 `bfd_cache_close`

```
bool bfd_cache_close (bfd *abfd);                         [Function]
    Remove the BFD abfd from the cache. If the attached file is open, then close it too.
    FALSE is returned if closing the file fails, TRUE is returned if all is well.
```

2.14.1.3 `bfd_cache_close_all`

```
bool bfd_cache_close_all (void);                          [Function]
    Remove all BFDs from the cache. If the attached file is open, then close it too.
    FALSE is returned if closing one of the file fails, TRUE is returned if all is well.
```

2.14.1.4 `bfd_open_file`

```
FILE* bfd_open_file (bfd *abfd);                         [Function]
    Call the OS to open a file for abfd. Return the FILE * (possibly NULL) that results from this operation. Set up the BFD so that future accesses know the file is open. If the FILE * returned is NULL, then it won't have been put in the cache, so it won't have to be removed from it.
```

2.15 Linker Functions

The linker uses three special entry points in the BFD target vector. It is not necessary to write special routines for these entry points when creating a new BFD back end, since generic versions are provided. However, writing them can speed up linking and make it use significantly less runtime memory.

The first routine creates a hash table used by the other routines. The second routine adds the symbols from an object file to the hash table. The third routine takes all the object files and links them together to create the output file. These routines are designed so that the linker proper does not need to know anything about the symbols in the object files that it is linking. The linker merely arranges the sections as directed by the linker script and lets BFD handle the details of symbols and relocs.

The second routine and third routines are passed a pointer to a `struct bfd_link_info` structure (defined in `bfdlink.h`) which holds information relevant to the link, including the linker hash table (which was created by the first routine) and a set of callback functions to the linker proper.

The generic linker routines are in `linker.c`, and use the header file `genlink.h`. As of this writing, the only back ends which have implemented versions of these routines are a.out (in `aoutx.h`) and ECOFF (in `ecoff.c`). The a.out routines are used as examples throughout this section.

2.15.1 Creating a linker hash table

The linker routines must create a hash table, which must be derived from `struct bfd_link_hash_table` described in `bfdlink.c`. See Section 2.16 [Hash Tables], page 179, for information on how to create a derived hash table. This entry point is called using the target vector of the linker output file.

The `_bfd_link_hash_table_create` entry point must allocate and initialize an instance of the desired hash table. If the back end does not require any additional information to be stored with the entries in the hash table, the entry point may simply create a `struct bfd_link_hash_table`. Most likely, however, some additional information will be needed.

For example, with each entry in the hash table the a.out linker keeps the index the symbol has in the final output file (this index number is used so that when doing a relocatable link the symbol index used in the output file can be quickly filled in when copying over a reloc). The a.out linker code defines the required structures and functions for a hash table derived from `struct bfd_link_hash_table`. The a.out linker hash table is created by the function `NAME(aout,link_hash_table_create)`; it simply allocates space for the hash table, initializes it, and returns a pointer to it.

When writing the linker routines for a new back end, you will generally not know exactly which fields will be required until you have finished. You should simply create a new hash table which defines no additional fields, and then simply add fields as they become necessary.

2.15.2 Adding symbols to the hash table

The linker proper will call the `_bfd_link_add_symbols` entry point for each object file or archive which is to be linked (typically these are the files named on the command line, but some may also come from the linker script). The entry point is responsible for examining the file. For an object file, BFD must add any relevant symbol information to the hash

table. For an archive, BFD must determine which elements of the archive should be used and adding them to the link.

The a.out version of this entry point is `NAME(aout,link_add_symbols)`.

2.15.2.1 Differing file formats

Normally all the files involved in a link will be of the same format, but it is also possible to link together different format object files, and the back end must support that. The `_bfd_link_add_symbols` entry point is called via the target vector of the file to be added. This has an important consequence: the function may not assume that the hash table is the type created by the corresponding `_bfd_link_hash_table_create` vector. All the `_bfd_link_add_symbols` function can assume about the hash table is that it is derived from `struct bfd_link_hash_table`.

Sometimes the `_bfd_link_add_symbols` function must store some information in the hash table entry to be used by the `_bfd_final_link` function. In such a case the output bfd xvec must be checked to make sure that the hash table was created by an object file of the same format.

The `_bfd_final_link` routine must be prepared to handle a hash entry without any extra information added by the `_bfd_link_add_symbols` function. A hash entry without extra information will also occur when the linker script directs the linker to create a symbol. Note that, regardless of how a hash table entry is added, all the fields will be initialized to some sort of null value by the hash table entry initialization function.

See `ecoff_link_add externals` for an example of how to check the output bfd before saving information (in this case, the ECOFF external symbol debugging information) in a hash table entry.

2.15.2.2 Adding symbols from an object file

When the `_bfd_link_add_symbols` routine is passed an object file, it must add all externally visible symbols in that object file to the hash table. The actual work of adding the symbol to the hash table is normally handled by the function `_bfd_generic_link_add_one_symbol`. The `_bfd_link_add_symbols` routine is responsible for reading all the symbols from the object file and passing the correct information to `_bfd_generic_link_add_one_symbol`.

The `_bfd_link_add_symbols` routine should not use `bfd_canonicalize_syntab` to read the symbols. The point of providing this routine is to avoid the overhead of converting the symbols into generic `asymbol` structures.

`_bfd_generic_link_add_one_symbol` handles the details of combining common symbols, warning about multiple definitions, and so forth. It takes arguments which describe the symbol to add, notably symbol flags, a section, and an offset. The symbol flags include such things as `BSF_WEAK` or `BSF_INDIRECT`. The section is a section in the object file, or something like `bfd_und_section_ptr` for an undefined symbol or `bfd_com_section_ptr` for a common symbol.

If the `_bfd_final_link` routine is also going to need to read the symbol information, the `_bfd_link_add_symbols` routine should save it somewhere attached to the object file BFD. However, the information should only be saved if the `keep_memory` field of the `info` argument is `TRUE`, so that the `-no-keep-memory` linker switch is effective.

The `a.out` function which adds symbols from an object file is `aout_link_add_object_symbols`, and most of the interesting work is in `aout_link_add_symbols`. The latter saves pointers to the hash table entries created by `_bfd_generic_link_add_one_symbol` indexed by symbol number, so that the `_bfd_final_link` routine does not have to call the hash table lookup routine to locate the entry.

2.15.2.3 Adding symbols from an archive

When the `_bfd_link_add_symbols` routine is passed an archive, it must look through the symbols defined by the archive and decide which elements of the archive should be included in the link. For each such element it must call the `add_archive_element` linker callback, and it must add the symbols from the object file to the linker hash table. (The callback may in fact indicate that a replacement BFD should be used, in which case the symbols from that BFD should be added to the linker hash table instead.)

In most cases the work of looking through the symbols in the archive should be done by the `_bfd_generic_link_add_archive_symbols` function. `_bfd_generic_link_add_archive_symbols` is passed a function to call to make the final decision about adding an archive element to the link and to do the actual work of adding the symbols to the linker hash table. If the element is to be included, the `add_archive_element` linker callback routine must be called with the element as an argument, and the element's symbols must be added to the linker hash table just as though the element had itself been passed to the `_bfd_link_add_symbols` function.

When the `a.out` `_bfd_link_add_symbols` function receives an archive, it calls `_bfd_generic_link_add_archive_symbols` passing `aout_link_check_archive_element` as the function argument. `aout_link_check_archive_element` calls `aout_link_check_ar_symbols`. If the latter decides to add the element (an element is only added if it provides a real, non-common, definition for a previously undefined or common symbol) it calls the `add_archive_element` callback and then `aout_link_check_archive_element` calls `aout_link_add_symbols` to actually add the symbols to the linker hash table - possibly those of a substitute BFD, if the `add_archive_element` callback avails itself of that option. The ECOFF back end is unusual in that it does not normally call `_bfd_generic_link_add_archive_symbols`, because ECOFF archives already contain a hash table of symbols. The ECOFF back end searches the archive itself to avoid the overhead of creating a new hash table.

2.15.3 Performing the final link

When all the input files have been processed, the linker calls the `_bfd_final_link` entry point of the output BFD. This routine is responsible for producing the final output file, which has several aspects. It must relocate the contents of the input sections and copy the data into the output sections. It must build an output symbol table including any local symbols from the input files and the global symbols from the hash table. When producing relocatable output, it must modify the input relocs and write them into the output file. There may also be object format dependent work to be done.

The linker will also call the `write_object_contents` entry point when the BFD is closed. The two entry points must work together in order to produce the correct output file.

The details of how this works are inevitably dependent upon the specific object file format. The `a.out` `_bfd_final_link` routine is `NAME(aout,final_link)`.

2.15.3.1 Information provided by the linker

Before the linker calls the `_bfd_final_link` entry point, it sets up some data structures for the function to use.

The `input_bfds` field of the `bfd_link_info` structure will point to a list of all the input files included in the link. These files are linked through the `link.next` field of the `bfd` structure.

Each section in the output file will have a list of `link_order` structures attached to the `map_head.link_order` field (the `link_order` structure is defined in `bfdlink.h`). These structures describe how to create the contents of the output section in terms of the contents of various input sections, fill constants, and, eventually, other types of information. They also describe relocs that must be created by the BFD backend, but do not correspond to any input file; this is used to support `-Ur`, which builds constructors while generating a relocatable object file.

2.15.3.2 Relocating the section contents

The `_bfd_final_link` function should look through the `link_order` structures attached to each section of the output file. Each `link_order` structure should either be handled specially, or it should be passed to the function `_bfd_default_link_order` which will do the right thing (`_bfd_default_link_order` is defined in `linker.c`).

For efficiency, a `link_order` of type `bfd_indirect_link_order` whose associated section belongs to a BFD of the same format as the output BFD must be handled specially. This type of `link_order` describes part of an output section in terms of a section belonging to one of the input files. The `_bfd_final_link` function should read the contents of the section and any associated relocs, apply the relocs to the section contents, and write out the modified section contents. If performing a relocatable link, the relocs themselves must also be modified and written out.

The functions `_bfd_relocate_contents` and `_bfd_final_link_relocate` provide some general support for performing the actual relocations, notably overflow checking. Their arguments include information about the symbol the relocation is against and a `reloc_howto_type` argument which describes the relocation to perform. These functions are defined in `reloc.c`.

The `a.out` function which handles reading, relocating, and writing section contents is `aout_link_input_section`. The actual relocation is done in `aout_link_input_section_std` and `aout_link_input_section_ext`.

2.15.3.3 Writing the symbol table

The `_bfd_final_link` function must gather all the symbols in the input files and write them out. It must also write out all the symbols in the global hash table. This must be controlled by the `strip` and `discard` fields of the `bfd_link_info` structure.

The local symbols of the input files will not have been entered into the linker hash table. The `_bfd_final_link` routine must consider each input file and include the symbols in the output file. It may be convenient to do this when looking through the `link_order` structures, or it may be done by stepping through the `input_bfds` list.

The `_bfd_final_link` routine must also traverse the global hash table to gather all the externally visible symbols. It is possible that most of the externally visible symbols may

be written out when considering the symbols of each input file, but it is still necessary to traverse the hash table since the linker script may have defined some symbols that are not in any of the input files.

The `strip` field of the `bfd_link_info` structure controls which symbols are written out. The possible values are listed in `bfdlink.h`. If the value is `strip_some`, then the `keep_hash` field of the `bfd_link_info` structure is a hash table of symbols to keep; each symbol should be looked up in this hash table, and only symbols which are present should be included in the output file.

If the `strip` field of the `bfd_link_info` structure permits local symbols to be written out, the `discard` field is used to further controls which local symbols are included in the output file. If the value is `discard_1`, then all local symbols which begin with a certain prefix are discarded; this is controlled by the `bfd_is_local_label_name` entry point.

The a.out backend handles symbols by calling `aout_link_write_symbols` on each input BFD and then traversing the global hash table with the function `aout_link_write_other_symbol`. It builds a string table while writing out the symbols, which is written to the output file at the end of `NAME(aout,final_link)`.

2.15.3.4 `bfd_link_split_section`

`bool bfd_link_split_section (bfd *abfd, asection *sec);` [Function]
Return nonzero if `sec` should be split during a reloceatable or final link.

```
#define bfd_link_split_section(abfd, sec) \
    BFD_SEND (abfd, _bfd_link_split_section, (abfd, sec))
```

2.15.3.5 `bfd_section_already_linked`

`bool bfd_section_already_linked (bfd *abfd, asection *sec, struct bfd_link_info *info);` [Function]
Check if `data` has been already linked during a reloceatable or final link. Return TRUE if it has.

```
#define bfd_section_already_linked(abfd, sec, info) \
    BFD_SEND (abfd, _section_already_linked, (abfd, sec, info))
```

2.15.3.6 `bfd_generic_define_common_symbol`

`bool bfd_generic_define_common_symbol (bfd *output_bfd, struct bfd_link_info *info, struct bfd_link_hash_entry *h);` [Function]
Convert common symbol `h` into a defined symbol. Return TRUE on success and FALSE on failure.

```
#define bfd_define_common_symbol(output_bfd, info, h) \
    BFD_SEND (output_bfd, _bfd_define_common_symbol, (output_bfd, info, h))
```

2.15.3.7 `_bfd_generic_link_hide_symbol`

```
void _bfd_generic_link_hide_symbol (bfd *output_bfd,          [Function]
                                   struct bfd_link_info *info, struct bfd_link_hash_entry *h);
```

Hide symbol *h*. This is an internal function. It should not be called from outside the BFD library.

```
#define bfd_link_hide_symbol(output_bfd, info, h) \
    BFD_SEND (output_bfd, _bfd_link_hide_symbol, (output_bfd, info, h))■
```

2.15.3.8 `bfd_generic_define_start_stop`

```
struct bfd_link_hash_entry *bfd_generic_define_start_stop [Function]
    (struct bfd_link_info *info, const char *symbol, asection
     *sec);
```

Define a `__start`, `__stop`, `.startof`. or `.sizeof`. symbol. Return the symbol or NULL if no such undefined symbol exists.

```
#define bfd_define_start_stop(output_bfd, info, symbol, sec) \
    BFD_SEND (output_bfd, _bfd_define_start_stop, (info, symbol, sec))■
```

2.15.3.9 `bfd_find_version_for_sym`

```
struct bfd_elf_version_tree * bfd_find_version_for_sym [Function]
    (struct bfd_elf_version_tree *verdefs, const char *sym_name,
     bool *hide);
```

Search an elf version script tree for symbol versioning info and export / don't-export status for a given symbol. Return non-NULL on success and NULL on failure; also sets the output 'hide' boolean parameter.

2.15.3.10 `bfd_hide_sym_by_version`

```
bool bfd_hide_sym_by_version (struct bfd_elf_version_tree [Function]
                              *verdefs, const char *sym_name);
```

Search an elf version script tree for symbol versioning info for a given symbol. Return TRUE if the symbol is hidden.

2.15.3.11 `bfd_link_check_relocs`

```
bool bfd_link_check_relocs (bfd *abfd, struct [Function]
                            bfd_link_info *info);
```

Checks the relocs in ABFD for validity. Does not execute the relocs. Return TRUE if everything is OK, FALSE otherwise. This is the external entry point to this code.

2.15.3.12 `_bfd_generic_link_check_relocs`

```
bool _bfd_generic_link_check_relocs (bfd *abfd, struct [Function]
                                      bfd_link_info *info);
```

Stub function for targets that do not implement reloc checking. Return TRUE. This is an internal function. It should not be called from outside the BFD library.

2.15.3.13 bfd_merge_private_bfd_data

`bool bfd_merge_private_bfd_data (bfd *ibfd, struct bfd_link_info *info);` [Function]

Merge private BFD information from the BFD *ibfd* to the the output file BFD when linking. Return TRUE on success, FALSE on error. Possible error returns are:

- `bfd_error_no_memory` - Not enough memory exists to create private data for *obfd*.

```
#define bfd_merge_private_bfd_data(ibfd, info) \
    BFD_SEND ((info)->output_bfd, _bfd_merge_private_bfd_data, \
              (ibfd, info))
```

2.15.3.14 _bfd_generic_verify_endian_match

`bool _bfd_generic_verify_endian_match (bfd *ibfd, struct bfd_link_info *info);` [Function]

Can be used from / for `bfd_merge_private_bfd_data` to check that endianness matches between input and output file. Returns TRUE for a match, otherwise returns FALSE and emits an error.

2.16 Hash Tables

BFD provides a simple set of hash table functions. Routines are provided to initialize a hash table, to free a hash table, to look up a string in a hash table and optionally create an entry for it, and to traverse a hash table. There is currently no routine to delete a string from a hash table.

The basic hash table does not permit any data to be stored with a string. However, a hash table is designed to present a base class from which other types of hash tables may be derived. These derived types may store additional information with the string. Hash tables were implemented in this way, rather than simply providing a data pointer in a hash table entry, because they were designed for use by the linker back ends. The linker may create thousands of hash table entries, and the overhead of allocating private data and storing and following pointers becomes noticeable.

The basic hash table code is in `hash.c`.

2.16.1 Creating and freeing a hash table

To create a hash table, create an instance of a `struct bfd_hash_table` (defined in `bfd.h`) and call `bfd_hash_table_init` (if you know approximately how many entries you will need, the function `bfd_hash_table_init_n`, which takes a *size* argument, may be used). `bfd_hash_table_init` returns FALSE if some sort of error occurs.

The function `bfd_hash_table_init` take as an argument a function to use to create new entries. For a basic hash table, use the function `bfd_hash_newfunc`. See Section 2.16.4 [Deriving a New Hash Table Type], page 180, for why you would want to use a different value for this argument.

`bfd_hash_table_init` will create an `objalloc` which will be used to allocate new entries. You may allocate memory on this `objalloc` using `bfd_hash_allocate`.

Use `bfd_hash_table_free` to free up all the memory that has been allocated for a hash table. This will not free up the `struct bfd_hash_table` itself, which you must provide.

Use `bfd_hash_set_default_size` to set the default size of hash table to use.

2.16.2 Looking up or entering a string

The function `bfd_hash_lookup` is used both to look up a string in the hash table and to create a new entry.

If the *create* argument is `FALSE`, `bfd_hash_lookup` will look up a string. If the string is found, it will return a pointer to a `struct bfd_hash_entry`. If the string is not found in the table `bfd_hash_lookup` will return `NULL`. You should not modify any of the fields in the returned `struct bfd_hash_entry`.

If the *create* argument is `TRUE`, the string will be entered into the hash table if it is not already there. Either way a pointer to a `struct bfd_hash_entry` will be returned, either to the existing structure or to a newly created one. In this case, a `NULL` return means that an error occurred.

If the *create* argument is `TRUE`, and a new entry is created, the *copy* argument is used to decide whether to copy the string onto the hash table object or not. If *copy* is passed as `FALSE`, you must be careful not to deallocate or modify the string as long as the hash table exists.

2.16.3 Traversing a hash table

The function `bfd_hash_traverse` may be used to traverse a hash table, calling a function on each element. The traversal is done in a random order.

`bfd_hash_traverse` takes as arguments a function and a generic `void *` pointer. The function is called with a hash table entry (a `struct bfd_hash_entry *`) and the generic pointer passed to `bfd_hash_traverse`. The function must return a `boolean` value, which indicates whether to continue traversing the hash table. If the function returns `FALSE`, `bfd_hash_traverse` will stop the traversal and return immediately.

2.16.4 Deriving a new hash table type

Many uses of hash tables want to store additional information which each entry in the hash table. Some also find it convenient to store additional information with the hash table itself. This may be done using a derived hash table.

Since C is not an object oriented language, creating a derived hash table requires sticking together some boilerplate routines with a few differences specific to the type of hash table you want to create.

An example of a derived hash table is the linker hash table. The structures for this are defined in `bfdlink.h`. The functions are in `linker.c`.

You may also derive a hash table from an already derived hash table. For example, the a.out linker backend code uses a hash table derived from the linker hash table.

2.16.4.1 Define the derived structures

You must define a structure for an entry in the hash table, and a structure for the hash table itself.

The first field in the structure for an entry in the hash table must be of the type used for an entry in the hash table you are deriving from. If you are deriving from a basic hash table this is `struct bfd_hash_entry`, which is defined in `bfd.h`. The first field in the structure for the hash table itself must be of the type of the hash table you are deriving from itself. If you are deriving from a basic hash table, this is `struct bfd_hash_table`.

For example, the linker hash table defines `struct bfd_link_hash_entry` (in `bfdlink.h`). The first field, `root`, is of type `struct bfd_hash_entry`. Similarly, the first field in `struct bfd_link_hash_table`, `table`, is of type `struct bfd_hash_table`.

2.16.4.2 Write the derived creation routine

You must write a routine which will create and initialize an entry in the hash table. This routine is passed as the function argument to `bfd_hash_table_init`.

In order to permit other hash tables to be derived from the hash table you are creating, this routine must be written in a standard way.

The first argument to the creation routine is a pointer to a hash table entry. This may be `NULL`, in which case the routine should allocate the right amount of space. Otherwise the space has already been allocated by a hash table type derived from this one.

After allocating space, the creation routine must call the creation routine of the hash table type it is derived from, passing in a pointer to the space it just allocated. This will initialize any fields used by the base hash table.

Finally the creation routine must initialize any local fields for the new hash table type.

Here is a boilerplate example of a creation routine. *function_name* is the name of the routine. *entry_type* is the type of an entry in the hash table you are creating. *base_newfunc* is the name of the creation routine of the hash table type your hash table is derived from.

```

struct bfd_hash_entry *
function_name (struct bfd_hash_entry *entry,
              struct bfd_hash_table *table,
              const char *string)
{
    struct entry_type *ret = (entry_type *) entry;

    /* Allocate the structure if it has not already been allocated by a
       derived class. */
    if (ret == NULL)
    {
        ret = bfd_hash_allocate (table, sizeof (* ret));
        if (ret == NULL)
            return NULL;
    }

    /* Call the allocation method of the base class. */
    ret = ((entry_type *)
          base_newfunc ((struct bfd_hash_entry *) ret, table, string));

    /* Initialize the local fields here. */

```

```

    return (struct bfd_hash_entry *) ret;
}

```

The creation routine for the linker hash table, which is in `linker.c`, looks just like this example. *function_name* is `_bfd_link_hash_newfunc`. *entry_type* is `struct bfd_link_hash_entry`. *base_newfunc* is `bfd_hash_newfunc`, the creation routine for a basic hash table.

`_bfd_link_hash_newfunc` also initializes the local fields in a linker hash table entry: `type`, `written` and `next`.

2.16.4.3 Write other derived routines

You will want to write other routines for your new hash table, as well.

You will want an initialization routine which calls the initialization routine of the hash table you are deriving from and initializes any other local fields. For the linker hash table, this is `_bfd_link_hash_table_init` in `linker.c`.

You will want a lookup routine which calls the lookup routine of the hash table you are deriving from and casts the result. The linker hash table uses `bfd_link_hash_lookup` in `linker.c` (this actually takes an additional argument which it uses to decide how to return the looked up value).

You may want a traversal routine. This should just call the traversal routine of the hash table you are deriving from with appropriate casts. The linker hash table uses `bfd_link_hash_traverse` in `linker.c`.

These routines may simply be defined as macros. For example, the a.out backend linker hash table, which is derived from the linker hash table, uses macros for the lookup and traversal routines. These are `aout_link_hash_lookup` and `aout_link_hash_traverse` in `aoutx.h`.

2.16.4.4 bfd_hash_table_init_n

```

bool bfd_hash_table_init_n (struct bfd_hash_table *,           [Function]
                           struct bfd_hash_entry *(* newfunc*) (struct bfd_hash_entry
                           *, struct bfd_hash_table *, const char *), unsigned int
                           *entsize*, unsigned int *size*);

```

Create a new hash table, given a number of entries.

2.16.4.5 bfd_hash_table_init

```

bool bfd_hash_table_init (struct bfd_hash_table *, struct      [Function]
                         bfd_hash_entry *(* newfunc*) (struct bfd_hash_entry *,
                         struct bfd_hash_table *, const char *), unsigned int
                         *entsize*);

```

Create a new hash table with the default number of entries.

2.16.4.6 bfd_hash_table_free

```

void bfd_hash_table_free (struct bfd_hash_table *);           [Function]

```

Free a hash table.

2.16.4.7 bfd_hash_lookup

```
struct bfd_hash_entry *bfd_hash_lookup (struct          [Function]
    bfd_hash_table *, const char *, bool *create*, bool *copy*);
    Look up a string in a hash table.
```

2.16.4.8 bfd_hash_insert

```
struct bfd_hash_entry *bfd_hash_insert (struct          [Function]
    bfd_hash_table *, const char *, unsigned long *hash*);
    Insert an entry in a hash table.
```

2.16.4.9 bfd_hash_rename

```
void bfd_hash_rename (struct bfd_hash_table *, const char  [Function]
    *, struct bfd_hash_entry *);
    Rename an entry in a hash table.
```

2.16.4.10 bfd_hash_replace

```
void bfd_hash_replace (struct bfd_hash_table *, struct    [Function]
    bfd_hash_entry * *old*, struct bfd_hash_entry * *new*);
    Replace an entry in a hash table.
```

2.16.4.11 bfd_hash_allocate

```
void *bfd_hash_allocate (struct bfd_hash_table *, unsigned [Function]
    int *size*);
    Allocate space in a hash table.
```

2.16.4.12 bfd_hash_newfunc

```
struct bfd_hash_entry *bfd_hash_newfunc (struct          [Function]
    bfd_hash_entry *, struct bfd_hash_table *, const char *);
    Base method for creating a new hash table entry.
```

2.16.4.13 bfd_hash_traverse

```
void bfd_hash_traverse (struct bfd_hash_table *, bool (*) [Function]
    (struct bfd_hash_entry *, void *), void *);
    Traverse a hash table.
```

2.16.4.14 bfd_hash_set_default_size

```
unsigned int bfd_hash_set_default_size (unsigned int);    [Function]
    Set hash table default size.
```

2.16.4.15 _bfd_stringtab_init

```
struct bfd_stringtab_hash *_bfd_stringtab_init (void);   [Function]
    Create a new stringtab.
```


2.16.4.16 `_bfd_xcoff_stringtab_init`

```
struct bfd_stringtab_hash *_bfd_xcoff_stringtab_init (bool      [Function]
    *isxcoff64*);
```

Create a new stringtab in which the strings are output in the format used in the XCOFF .debug section: a two byte length precedes each string.

2.16.4.17 `_bfd_stringtab_free`

```
void _bfd_stringtab_free (struct bfd_stringtab_hash *);      [Function]
    Free a stringtab.
```

2.16.4.18 `_bfd_stringtab_add`

```
bfd_size_type _bfd_stringtab_add (struct bfd_stringtab_hash  [Function]
    *, const char *, bool *hash*, bool *copy*);
```

Get the index of a string in a stringtab, adding it if it is not already present. If HASH is FALSE, we don't really use the hash table, and we don't eliminate duplicate strings. If COPY is true then store a copy of STR if creating a new entry.

2.16.4.19 `_bfd_stringtab_size`

```
bfd_size_type _bfd_stringtab_size (struct bfd_stringtab_hash  [Function]
    *);
    Get the number of bytes in a stringtab.
```

2.16.4.20 `_bfd_stringtab_emit`

```
bool _bfd_stringtab_emit (bfd *, struct bfd_stringtab_hash *); [Function]
    Write out a stringtab. ABFD must already be at the right location in the file.
```

3 BFD back ends

3.1 What to Put Where

All of BFD lives in one directory.

3.2 a.out backends

BFD supports a number of different flavours of a.out format, though the major differences are only the sizes of the structures on disk, and the shape of the relocation information.

The support is split into a basic support file `aoutx.h` and other files which derive functions from the base. One derivation file is `aoutf1.h` (for a.out flavour 1), and adds to the basic a.out functions support for sun3, sun4, and 386 a.out files, to create a target jump vector for a specific target.

This information is further split out into more specific files for each machine, including `sunos.c` for sun3 and sun4, and `demo64.c` for a demonstration of a 64 bit a.out format.

The base file `aoutx.h` defines general mechanisms for reading and writing records to and from disk and various other methods which BFD requires. It is included by `aout32.c` and `aout64.c` to form the names `aout_32_swap_exec_header_in`, `aout_64_swap_exec_header_in`, etc.

As an example, this is what goes on to make the back end for a sun4, from `aout32.c`:

```
#define ARCH_SIZE 32
#include "aoutx.h"
```

Which exports names:

```
...
aout_32_canonicalize_reloc
aout_32_find_nearest_line
aout_32_get_lineno
aout_32_get_reloc_upper_bound
...
```

from `sunos.c`:

```
#define TARGET_NAME "a.out-sunos-big"
#define VECNAME      sparc_aout_sunos_be_vec
#include "aoutf1.h"
```

requires all the names from `aout32.c`, and produces the jump vector

```
sparc_aout_sunos_be_vec
```

The file `host-aout.c` is a special case. It is for a large set of hosts that use “more or less standard” a.out files, and for which cross-debugging is not interesting. It uses the standard 32-bit a.out support routines, but determines the file offsets and addresses of the text, data, and BSS sections, the machine architecture and machine type, and the entry point address, in a host-dependent manner. Once these values have been determined, generic code is used to handle the object file.

When porting it to run on a new system, you must supply:

```
HOST_PAGE_SIZE
```

```

HOST_SEGMENT_SIZE
HOST_MACHINE_ARCH      (optional)
HOST_MACHINE_MACHINE   (optional)
HOST_TEXT_START_ADDR
HOST_STACK_END_ADDR

```

in the file `../include/sys/h-XXX.h` (for your host). These values, plus the structures and macros defined in `a.out.h` on your host system, will produce a BFD target that will access ordinary `a.out` files on your host. To configure a new machine to use `host-aout.c`, specify:

```

TDEFAULTS = -DDEFAULT_VECTOR=host_aout_big_vec
TDEPFILES= host-aout.o trad-core.o

```

in the `config/XXX.mt` file, and modify `configure.ac` to use the `XXX.mt` file (by setting `"bfd_target=XXX"`) when your configuration is selected.

3.2.1 Relocations

The file `aoutx.h` provides for both the *standard* and *extended* forms of `a.out` relocation records.

The standard records contain only an address, a symbol index, and a type field. The extended records also have a full integer for an addend.

3.2.2 Internal entry points

`aoutx.h` exports several routines for accessing the contents of an `a.out` file, which are gathered and exported in turn by various format specific files (eg `sunos.c`).

3.2.2.1 `aout_size_swap_exec_header_in`

```

void aout_size_swap_exec_header_in, (bfd *abfd, struct      [Function]
    external_exec *bytes, struct internal_exec *execp);

```

Swap the information in an executable header `raw_bytes` taken from a raw byte stream memory image into the internal exec header structure `execp`.

3.2.2.2 `aout_size_swap_exec_header_out`

```

void aout_size_swap_exec_header_out (bfd *abfd, struct      [Function]
    internal_exec *execp, struct external_exec *raw_bytes);

```

Swap the information in an internal exec header structure `execp` into the buffer `raw_bytes` ready for writing to disk.

3.2.2.3 `aout_size_some_aout_object_p`

```

bfd_cleanup aout_size_some_aout_object_p (bfd *abfd,        [Function]
    struct internal_exec *execp, bfd_cleanup
    (*callback_to_real_object_p) (bfd *));

```

Some `a.out` variant thinks that the file open in `abfd` checking is an `a.out` file. Do some more checking, and set up for access if it really is. Call back to the calling environment's "finish up" function just before returning, to handle any last-minute setup.

3.2.2.4 `aout_size_mkobject`

`bool aout_size_mkobject, (bfd *abfd);` [Function]
 Initialize BFD *abfd* for use with a.out files.

3.2.2.5 `aout_size_machine_type`

`enum machine_type aout_size_machine_type (enum` [Function]
`bfd_architecture arch, unsigned long machine, bool *unknown);`
 Keep track of machine architecture and machine type for a.out's. Return the `machine_type` for a particular architecture and machine, or `M_UNKNOWN` if that exact architecture and machine can't be represented in a.out format.

If the architecture is understood, machine type 0 (default) is always understood.

3.2.2.6 `aout_size_set_arch_mach`

`bool aout_size_set_arch_mach, (bfd *, enum` [Function]
`bfd_architecture arch, unsigned long machine);`
 Set the architecture and the machine of the BFD *abfd* to the values *arch* and *machine*. Verify that *abfd*'s format can support the architecture required.

3.2.2.7 `aout_size_new_section_hook`

`bool aout_size_new_section_hook, (bfd *abfd, asection` [Function]
`*newsect);`
 Called by the BFD in response to a `bfd_make_section` request.

3.3 coff backends

BFD supports a number of different flavours of coff format. The major differences between formats are the sizes and alignments of fields in structures on disk, and the occasional extra field.

Coff in all its varieties is implemented with a few common files and a number of implementation specific files. For example, the i386 coff format is implemented in the file `coff-i386.c`. This file `#includes` `coff/i386.h` which defines the external structure of the coff format for the i386, and `coff/internal.h` which defines the internal structure. `coff-i386.c` also defines the relocations used by the i386 coff format See Section 2.9 [Relocations], page 48.

3.3.1 Porting to a new version of coff

The recommended method is to select from the existing implementations the version of coff which is most like the one you want to use. For example, we'll say that i386 coff is the one you select, and that your coff flavour is called foo. Copy `i386coff.c` to `foocoff.c`, copy `../include/coff/i386.h` to `../include/coff/foo.h`, and add the lines to `targets.c` and `Makefile.in` so that your new back end is used. Alter the shapes of the structures in `../include/coff/foo.h` so that they match what you need. You will probably also have to add `#ifdefs` to the code in `coff/internal.h` and `coffcode.h` if your version of coff is too wild.

You can verify that your new BFD backend works quite simply by building `objdump` from the `binutils` directory, and making sure that its version of what's going on and your host system's idea (assuming it has the pretty standard `coff dump` utility, usually called `att-dump` or just `dump`) are the same. Then clean up your code, and send what you've done to Cygnus. Then your stuff will be in the next release, and you won't have to keep integrating it.

3.3.2 How the `coff` backend works

3.3.2.1 File layout

The `Coff` backend is split into generic routines that are applicable to any `Coff` target and routines that are specific to a particular target. The target-specific routines are further split into ones which are basically the same for all `Coff` targets except that they use the external symbol format or use different values for certain constants.

The generic routines are in `coffgen.c`. These routines work for any `Coff` target. They use some hooks into the target specific code; the hooks are in a `bfd_coff_backend_data` structure, one of which exists for each target.

The essentially similar target-specific routines are in `coffcode.h`. This header file includes executable C code. The various `Coff` targets first include the appropriate `Coff` header file, make any special defines that are needed, and then include `coffcode.h`.

Some of the `Coff` targets then also have additional routines in the target source file itself.

3.3.2.2 `Coff` long section names

In the standard `Coff` object format, section names are limited to the eight bytes available in the `s_name` field of the `SCNHDR` section header structure. The format requires the field to be NUL-padded, but not necessarily NUL-terminated, so the longest section names permitted are a full eight characters.

The Microsoft PE variants of the `Coff` object file format add an extension to support the use of long section names. This extension is defined in section 4 of the Microsoft PE/COFF specification (rev 8.1). If a section name is too long to fit into the section header's `s_name` field, it is instead placed into the string table, and the `s_name` field is filled with a slash ("/") followed by the ASCII decimal representation of the offset of the full name relative to the string table base.

Note that this implies that the extension can only be used in object files, as executables do not contain a string table. The standard specifies that long section names from objects emitted into executable images are to be truncated.

However, as a GNU extension, BFD can generate executable images that contain a string table and long section names. This would appear to be technically valid, as the standard only says that `Coff` debugging information is deprecated, not forbidden, and in practice it works, although some tools that parse PE files expecting the MS standard format may become confused; `PEview` is one known example.

The functionality is supported in BFD by code implemented under the control of the macro `COFF_LONG_SECTION_NAMES`. If not defined, the format does not support long section names in any way. If defined, it is used to initialise a flag, `_bfd_coff_long_section_names`, and a hook function pointer, `_bfd_coff_set_long_section_names`, in the `Coff` backend data structure. The flag controls the generation of long section names in output BFDs at runtime;

if it is false, as it will be by default when generating an executable image, long section names are truncated; if true, the long section names extension is employed. The hook points to a function that allows the value of a copy of the flag in coff object tdata to be altered at runtime, on formats that support long section names at all; on other formats it points to a stub that returns an error indication.

With input BFDs, the flag is set according to whether any long section names are detected while reading the section headers. For a completely new BFD, the flag is set to the default for the target format. This information can be used by a client of the BFD library when deciding what output format to generate, and means that a BFD that is opened for read and subsequently converted to a writeable BFD and modified in-place will retain whatever format it had on input.

If `COFF_LONG_SECTION_NAMES` is simply defined (blank), or is defined to the value "1", then long section names are enabled by default; if it is defined to the value zero, they are disabled by default (but still accepted in input BFDs). The header `coffcode.h` defines a macro, `COFF_DEFAULT_LONG_SECTION_NAMES`, which is used in the backends to initialise the backend data structure fields appropriately; see the comments for further detail.

3.3.2.3 Bit twiddling

Each flavour of coff supported in BFD has its own header file describing the external layout of the structures. There is also an internal description of the coff layout, in `coff/internal.h`. A major function of the coff backend is swapping the bytes and twiddling the bits to translate the external form of the structures into the normal internal form. This is all performed in the `bfd_swap_thing_direction` routines. Some elements are different sizes between different versions of coff; it is the duty of the coff version specific include file to override the definitions of various packing routines in `coffcode.h`. E.g., the size of line number entry in coff is sometimes 16 bits, and sometimes 32 bits. `#define`ing `PUT_LNSZ_LNNO` and `GET_LNSZ_LNNO` will select the correct one. No doubt, some day someone will find a version of coff which has a varying field size not catered to at the moment. To port BFD, that person will have to add more `#defines`. Three of the bit twiddling routines are exported to `gdb`; `coff_swap_aux_in`, `coff_swap_sym_in` and `coff_swap_lineno_in`. `GDB` reads the symbol table on its own, but uses BFD to fix things up. More of the bit twiddlers are exported for `gas`; `coff_swap_aux_out`, `coff_swap_sym_out`, `coff_swap_lineno_out`, `coff_swap_reloc_out`, `coff_swap_filehdr_out`, `coff_swap_aouthdr_out`, `coff_swap_scnhdr_out`. `Gas` currently keeps track of all the symbol table and reloc drudgery itself, thereby saving the internal BFD overhead, but uses BFD to swap things on the way out, making cross ports much safer. Doing so also allows BFD (and thus the linker) to use the same header files as `gas`, which makes one avenue to disaster disappear.

3.3.2.4 Symbol reading

The simple canonical form for symbols used by BFD is not rich enough to keep all the information available in a coff symbol table. The back end gets around this problem by keeping the original symbol table around, "behind the scenes".

When a symbol table is requested (through a call to `bfd_canonicalize_symtab`), a request gets through to `coff_get_normalized_symtab`. This reads the symbol table from the coff file and swaps all the structures inside into the internal form. It also fixes up all the pointers in the table (represented in the file by offsets from the first symbol in the table) into physical

pointers to elements in the new internal table. This involves some work since the meanings of fields change depending upon context: a field that is a pointer to another structure in the symbol table at one moment may be the size in bytes of a structure at the next. Another pass is made over the table. All symbols which mark file names (`C_FILE` symbols) are modified so that the internal string points to the value in the auxent (the real filename) rather than the normal text associated with the symbol (`".file"`).

At this time the symbol names are moved around. Coff stores all symbols less than nine characters long physically within the symbol table; longer strings are kept at the end of the file in the string table. This pass moves all strings into memory and replaces them with pointers to the strings.

The symbol table is massaged once again, this time to create the canonical table used by the BFD application. Each symbol is inspected in turn, and a decision made (using the `sclass` field) about the various flags to set in the `asymbol`. See Section 2.6 [Symbols], page 38. The generated canonical table shares strings with the hidden internal symbol table.

Any linenumbers are read from the coff file too, and attached to the symbols which own the functions the linenumbers belong to.

3.3.2.5 Symbol writing

Writing a symbol to a coff file which didn't come from a coff file will lose any debugging information. The `asymbol` structure remembers the BFD from which the symbol was taken, and on output the back end makes sure that the same destination target as source target is present.

When the symbols have come from a coff file then all the debugging information is preserved. Symbol tables are provided for writing to the back end in a vector of pointers to pointers. This allows applications like the linker to accumulate and output large symbol tables without having to do too much byte copying.

This function runs through the provided symbol table and patches each symbol marked as a file place holder (`C_FILE`) to point to the next file place holder in the list. It also marks each `offset` field in the list with the offset from the first symbol of the current symbol.

Another function of this procedure is to turn the canonical value form of BFD into the form used by coff. Internally, BFD expects symbol values to be offsets from a section base; so a symbol physically at 0x120, but in a section starting at 0x100, would have the value 0x20. Coff expects symbols to contain their final value, so symbols have their values changed at this point to reflect their sum with their owning section. This transformation uses the `output_section` field of the `asymbol`'s `asection`. See Section 2.5 [Sections], page 23.

- `coff_mangle_symbols`

This routine runs through the provided symbol table and uses the offsets generated by the previous pass and the pointers generated when the symbol table was read in to create the structured hierarchy required by coff. It changes each pointer to a symbol into the index into the symbol table of the `asymbol`.

- `coff_write_symbols`

This routine runs through the symbol table and patches up the symbols from their internal form into the coff way, calls the bit twiddlers, and writes out the table to the file.

3.3.2.6 coff_symbol_type

The hidden information for an `asymbol` is described in a `combined_entry_type`:

```
typedef struct coff_ptr_struct
{
    /* Remembers the offset from the first symbol in the file for
       this symbol. Generated by coff_renumber_symbols. */
    unsigned int offset;

    /* Selects between the elements of the union below. */
    unsigned int is_sym : 1;

    /* Selects between the elements of the x_sym.x_tagndx union. If set,
       p is valid and the field will be renumbered. */
    unsigned int fix_tag : 1;

    /* Selects between the elements of the x_sym.x_fcny.x_fcn.x_endndx
       union. If set, p is valid and the field will be renumbered. */
    unsigned int fix_end : 1;

    /* Selects between the elements of the x_csect.x_scnlen union. If set,
       p is valid and the field will be renumbered. */
    unsigned int fix_scnlen : 1;

    /* If set, u.syment.n_value contains a pointer to a symbol. The final
       value will be the offset field. Used for XCOFF C_BSTAT symbols. */
    unsigned int fix_value : 1;

    /* If set, u.syment.n_value is an index into the line number entries.
       Used for XCOFF C_BINCL/C_EINCL symbols. */
    unsigned int fix_line : 1;

    /* The container for the symbol structure as read and translated
       from the file. */
    union
    {
        union internal_auxent auxent;
        struct internal_syment syment;
    } u;

    /* An extra pointer which can be used by format based on COFF (like XCOFF)
       to provide extra information to their backend. */
    void *extrap;
} combined_entry_type;

/* Each canonical asymbol really looks like this: */
```



```

typedef struct coff_symbol_struct
{
    /* The actual symbol which the rest of BFD works with */
    asymbol symbol;

    /* A pointer to the hidden information for this symbol */
    combined_entry_type *native;

    /* A pointer to the lineno information for this symbol */
    struct lineno_cache_entry *lineno;

    /* Have the line numbers been relocated yet ? */
    bool done_lineno;
} coff_symbol_type;

```

3.3.2.7 bfd_coff_backend_data

```

typedef struct
{
    void (*_bfd_coff_swap_aux_in)
        (bfd *, void *, int, int, int, int, void *);

    void (*_bfd_coff_swap_sym_in)
        (bfd *, void *, void *);

    void (*_bfd_coff_swap_lineno_in)
        (bfd *, void *, void *);

    unsigned int (*_bfd_coff_swap_aux_out)
        (bfd *, void *, int, int, int, int, void *);

    unsigned int (*_bfd_coff_swap_sym_out)
        (bfd *, void *, void *);

    unsigned int (*_bfd_coff_swap_lineno_out)
        (bfd *, void *, void *);

    unsigned int (*_bfd_coff_swap_reloc_out)
        (bfd *, void *, void *);

    unsigned int (*_bfd_coff_swap_filehdr_out)
        (bfd *, void *, void *);

    unsigned int (*_bfd_coff_swap_aouthdr_out)
        (bfd *, void *, void *);
}

```

```
unsigned int (*_bfd_coff_swap_scnhdr_out)
    (bfd *, void *, void *);

unsigned int _bfd_filhsz;
unsigned int _bfd_aoutsz;
unsigned int _bfd_scnhsz;
unsigned int _bfd_symesz;
unsigned int _bfd_auxesz;
unsigned int _bfd_relsz;
unsigned int _bfd_linesz;
unsigned int _bfd_filnmlen;
bool _bfd_coff_long_filenames;

bool _bfd_coff_long_section_names;
bool (*_bfd_coff_set_long_section_names)
    (bfd *, int);

unsigned int _bfd_coff_default_section_alignment_power;
bool _bfd_coff_force_symnames_in_strings;
unsigned int _bfd_coff_debug_string_prefix_length;
unsigned int _bfd_coff_max_nscns;

void (*_bfd_coff_swap_filehdr_in)
    (bfd *, void *, void *);

void (*_bfd_coff_swap_aouthdr_in)
    (bfd *, void *, void *);

void (*_bfd_coff_swap_scnhdr_in)
    (bfd *, void *, void *);

void (*_bfd_coff_swap_reloc_in)
    (bfd *abfd, void *, void *);

bool (*_bfd_coff_bad_format_hook)
    (bfd *, void *);

bool (*_bfd_coff_set_arch_mach_hook)
    (bfd *, void *);

void * (*_bfd_coff_mkobject_hook)
    (bfd *, void *, void *);

bool (*_bfd_styp_to_sec_flags_hook)
    (bfd *, void *, const char *, asection *, flagword *);

void (*_bfd_set_alignment_hook)
```

```
(bfd *, asection *, void *);

bool (*_bfd_coff_slurp_symbol_table)
(bfd *);

bool (*_bfd_coff_symname_in_debug)
(bfd *, struct internal_syment *);

bool (*_bfd_coff_pointerize_aux_hook)
(bfd *, combined_entry_type *, combined_entry_type *,
unsigned int, combined_entry_type *);

bool (*_bfd_coff_print_aux)
(bfd *, FILE *, combined_entry_type *, combined_entry_type *,
combined_entry_type *, unsigned int);

bool (*_bfd_coff_reloc16_extra_cases)
(bfd *, struct bfd_link_info *, struct bfd_link_order *, arelent *,
bfd_byte *, size_t *, size_t *);

int (*_bfd_coff_reloc16_estimate)
(bfd *, asection *, arelent *, unsigned int,
struct bfd_link_info *);

enum coff_symbol_classification (*_bfd_coff_classify_symbol)
(bfd *, struct internal_syment *);

bool (*_bfd_coff_compute_section_file_positions)
(bfd *);

bool (*_bfd_coff_start_final_link)
(bfd *, struct bfd_link_info *);

bool (*_bfd_coff_relocate_section)
(bfd *, struct bfd_link_info *, bfd *, asection *, bfd_byte *,
struct internal_reloc *, struct internal_syment *, asection **);

reloc_howto_type (*_bfd_coff_rtype_to_howto)
(bfd *, asection *, struct internal_reloc *,
struct coff_link_hash_entry *, struct internal_syment *, bfd_vma *);

bool (*_bfd_coff_adjust_symndx)
(bfd *, struct bfd_link_info *, bfd *, asection *,
struct internal_reloc *, bool *);

bool (*_bfd_coff_link_add_one_symbol)
(struct bfd_link_info *, bfd *, const char *, flagword,
```

```

    asection *, bfd_vma, const char *, bool, bool,
    struct bfd_link_hash_entry **);

bool (*_bfd_coff_link_output_has_begun)
    (bfd *, struct coff_final_link_info *);

bool (*_bfd_coff_final_link_postscript)
    (bfd *, struct coff_final_link_info *);

bool (*_bfd_coff_print_pdata)
    (bfd *, void *);

} bfd_coff_backend_data;

```

3.3.2.8 Writing relocations

To write relocations, the back end steps through the canonical relocation table and create an `internal_reloc`. The symbol index to use is removed from the `offset` field in the symbol table supplied. The address comes directly from the sum of the section base address and the relocation offset; the type is dug directly from the `howto` field. Then the `internal_reloc` is swapped into the shape of an `external_reloc` and written out to disk.

3.3.2.9 Reading linenumbers

Creating the linenumber table is done by reading in the entire coff linenumber table, and creating another table for internal use.

A coff linenumber table is structured so that each function is marked as having a line number of 0. Each line within the function is an offset from the first line in the function. The base of the line number information for the table is stored in the symbol associated with the function.

Note: The PE format uses line number 0 for a flag indicating a new source file.

The information is copied from the external to the internal table, and each symbol which marks a function is marked by pointing its...

How does this work ?

3.3.2.10 Reading relocations

Coff relocations are easily transformed into the internal BFD form (`arelent`).

Reading a coff relocation table is done in the following stages:

- Read the entire coff relocation table into memory.
- Process each relocation in turn; first swap it from the external to the internal form.
- Turn the symbol referenced in the relocation's symbol index into a pointer into the canonical symbol table. This table is the same as the one returned by a call to `bfd_canonicalize_symtab`. The back end will call that routine and save the result if a canonicalization hasn't been done.

- The reloc index is turned into a pointer to a howto structure, in a back end specific way. For instance, the 386 uses the `r_type` to directly produce an index into a howto table vector.
- Note that `arelent.addend` for COFF is often not what most people understand as a relocation addend, but rather an adjustment to the relocation addend stored in section contents of relocatable object files. The value found in section contents may also be confusing, depending on both symbol value and addend somewhat similar to the field value for a final-linked object. See `CALC_ADDEND`.

3.4 ELF backends

BFD support for ELF formats is being worked on. Currently, the best supported back ends are for sparc and i386 (running svr4 or Solaris 2).

Documentation of the internals of the support code still needs to be written. The code is changing quickly enough that we haven't bothered yet.

3.5 mmo backend

The mmo object format is used exclusively together with Professor Donald E. Knuth's educational 64-bit processor MMIX. The simulator `mmix` which is available at <http://mmix.cs.hm.edu/src/index.html> understands this format. That package also includes a combined assembler and linker called `mmixal`. The mmo format has no advantages feature-wise compared to e.g. ELF. It is a simple non-relocatable object format with no support for archives or debugging information, except for symbol value information and line numbers (which is not yet implemented in BFD). See <http://mmix.cs.hm.edu/> for more information about MMIX. The ELF format is used for intermediate object files in the BFD implementation.

3.5.1 File layout

The mmo file contents is not partitioned into named sections as with e.g. ELF. Memory areas is formed by specifying the location of the data that follows. Only the memory area '0x0000...00' to '0x01ff...ff' is executable, so it is used for code (and constants) and the area '0x2000...00' to '0x20ff...ff' is used for writable data. See Section 3.5.3 [mmo section mapping], page 200.

There is provision for specifying "special data" of 65536 different types. We use type 80 (decimal), arbitrarily chosen the same as the ELF `e_machine` number for MMIX, filling it with section information normally found in ELF objects. See Section 3.5.3 [mmo section mapping], page 200.

Contents is entered as 32-bit words, xor:ed over previous contents, always zero-initialized. A word that starts with the byte '0x98' forms a command called a 'lopcode', where the next byte distinguished between the thirteen lopcodes. The two remaining bytes, called the 'Y' and 'Z' fields, or the 'YZ' field (a 16-bit big-endian number), are used for various purposes different for each lopcode. As documented in <http://mmix.cs.hm.edu/doc/mmixal.pdf>, the lopcodes are:

lop_quote

0x98000001. The next word is contents, regardless of whether it starts with 0x98 or not.

- lop_loc** 0x9801YYZZ, where ‘Z’ is 1 or 2. This is a location directive, setting the location for the next data to the next 32-bit word (for $Z = 1$) or 64-bit word (for $Z = 2$), plus $Y * 2^{56}$. Normally ‘Y’ is 0 for the text segment and 2 for the data segment. Beware that the low bits of non-tetabyte-aligned values are silently discarded when being automatically incremented and when storing contents (in contrast to e.g. its use as current location when followed by `lop_fixo` et al before the next possibly-quoted tetabyte contents).
- lop_skip** 0x9802YYZZ. Increase the current location by ‘YZ’ bytes.
- lop_fixo** 0x9803YYZZ, where ‘Z’ is 1 or 2. Store the current location as 64 bits into the location pointed to by the next 32-bit ($Z = 1$) or 64-bit ($Z = 2$) word, plus $Y * 2^{56}$.
- lop_fixr** 0x9804YYZZ. ‘YZ’ is stored into the current location plus $2 - 4 * YZ$.
- lop_fixrx** 0x980500ZZ. ‘Z’ is 16 or 24. A value ‘L’ derived from the following 32-bit word are used in a manner similar to ‘YZ’ in `lop_fixr`: it is xor’ed into the current location minus $4 * L$. The first byte of the word is 0 or 1. If it is 1, then $L = (\text{lowest24bitsofword}) - 2^Z$, if 0, then $L = (\text{lowest24bitsofword})$.
- lop_file** 0x9806YYZZ. ‘Y’ is the file number, ‘Z’ is count of 32-bit words. Set the file number to ‘Y’ and the line counter to 0. The next $Z * 4$ bytes contain the file name, padded with zeros if the count is not a multiple of four. The same ‘Y’ may occur multiple times, but ‘Z’ must be 0 for all but the first occurrence.
- lop_line** 0x9807YYZZ. ‘YZ’ is the line number. Together with `lop_file`, it forms the source location for the next 32-bit word. Note that for each non-lopcode 32-bit word, line numbers are assumed incremented by one.
- lop_spec** 0x9808YYZZ. ‘YZ’ is the type number. Data until the next lopcode other than `lop_quote` forms special data of type ‘YZ’. See Section 3.5.3 [mmo section mapping], page 200.
Other types than 80, (or type 80 with a content that does not parse) is stored in sections named `.MMIX.spec_data.n` where n is the ‘YZ’-type. The flags for such a sections say not to allocate or load the data. The `vma` is 0. Contents of multiple occurrences of special data n is concatenated to the data of the previous `lop_spec ns`. The location in data or code at which the `lop_spec` occurred is lost.
- lop_pre** 0x980901ZZ. The first lopcode in a file. The ‘Z’ field forms the length of header information in 32-bit words, where the first word tells the time in seconds since ‘00:00:00 GMT Jan 1 1970’.
- lop_post** 0x980a00ZZ. $Z > 32$. This lopcode follows after all content-generating lopcodes in a program. The ‘Z’ field denotes the value of ‘rG’ at the beginning of the program. The following $256 - Z$ big-endian 64-bit words are loaded into global registers ‘\$G’ . . . ‘\$255’.
- lop_stab** 0x980b0000. The next-to-last lopcode in a program. Must follow immediately after the `lop_post` lopcode and its data. After this lopcode follows all symbols in a compressed format (see Section 3.5.2 [Symbol-table], page 198).

`lop_end` 0x980cYYZZ. The last lopcode in a program. It must follow the `lop_stab` lopcode and its data. The ‘YZ’ field contains the number of 32-bit words of symbol table information after the preceding `lop_stab` lopcode.

Note that the lopcode "fixups"; `lop_fixr`, `lop_fixrx` and `lop_fixo` are not generated by BFD, but are handled. They are generated by `mmixal`.

This trivial one-label, one-instruction file:

```
:Main TRAP 1,2,3
```

can be represented this way in mmo:

```
0x98090101 - lop_pre, one 32-bit word with timestamp.
<timestamp>
0x98010002 - lop_loc, text segment, using a 64-bit address.
    Note that mmixal does not emit this for the file above.
0x00000000 - Address, high 32 bits.
0x00000000 - Address, low 32 bits.
0x98060002 - lop_file, 2 32-bit words for file-name.
0x74657374 - "test"
0x2e730000 - ".s\0\0"
0x98070001 - lop_line, line 1.
0x00010203 - TRAP 1,2,3
0x980a00ff - lop_post, setting $255 to 0.
0x00000000
0x00000000
0x980b0000 - lop_stab for ":Main" = 0, serial 1.
0x203a4040 See Section 3.5.2 [Symbol-table], page 198.
0x10404020
0x4d206120
0x69016e00
0x81000000
0x980c0005 - lop_end; symbol table contained five 32-bit words.
```

3.5.2 Symbol table format

From `mmixal.w` (or really, the generated `mmixal.tex`) in the MMIXware package which also contains the `mmix` simulator: “Symbols are stored and retrieved by means of a ‘**ternary search trie**’, following ideas of Bentley and Sedgewick. (See ACM–SIAM Symp. on Discrete Algorithms ‘8’ (1997), 360–369; R.Sedgewick, ‘**Algorithms in C**’ (Reading, Mass. Addison–Wesley, 1998), ‘15.4’.) Each trie node stores a character, and there are branches to subtrees for the cases where a given character is less than, equal to, or greater than the character in the trie. There also is a pointer to a symbol table entry if a symbol ends at the current node.”

So it’s a tree encoded as a stream of bytes. The stream of bytes acts on a single virtual global symbol, adding and removing characters and signalling complete symbol points. Here, we read the stream and create symbols at the completion points.

First, there’s a control byte `m`. If any of the listed bits in `m` is nonzero, we execute what stands at the right, in the listed order:

```
(MM03_LEFT)
```

```

0x40 - Traverse left trie.
      (Read a new command byte and recurse.)

(MM03_SYMBITS)
0x2f - Read the next byte as a character and store it in the
      current character position; increment character position.
      Test the bits of m:

      (MM03_WCHAR)
0x80 - The character is 16-bit (so read another byte,
      merge into current character.

      (MM03_TYPEBITS)
0xf  - We have a complete symbol; parse the type, value
      and serial number and do what should be done
      with a symbol. The type and length information
      is in  $j = (m \& 0xf)$ .

      (MM03_REGQUAL_BITS)
j == 0xf: A register variable. The following
          byte tells which register.
j <= 8:  An absolute symbol. Read j bytes as the
          big-endian number the symbol equals.
          A  $j = 2$  with two zero bytes denotes an
          unknown symbol.
j > 8:   As with  $j \leq 8$ , but add  $(0x20 \ll 56)$ 
          to the value in the following  $j - 8$ 
          bytes.

      Then comes the serial number, as a variant of
      uleb128, but better named ubeb128:
      Read bytes and shift the previous value left 7
      (multiply by 128). Add in the new byte, repeat
      until a byte has bit 7 set. The serial number
      is the computed value minus 128.

      (MM03_MIDDLE)
0x20 - Traverse middle trie. (Read a new command byte
      and recurse.) Decrement character position.

      (MM03_RIGHT)
0x10 - Traverse right trie. (Read a new command byte and
      recurse.)

```

Let's look again at the `lop_stab` for the trivial file (see Section 3.5.1 [File layout], page 196).

```

0x980b0000 - lop_stab for ":Main" = 0, serial 1.
0x203a4040

```



```

0x10404020
0x4d206120
0x69016e00
0x81000000

```

This forms the trivial trie (note that the path between “:” and “M” is redundant):

```

203a      ":"
40        /
40        /
10        \
40        /
40        /
204d     "M"
2061     "a"
2069     "i"
016e     "n" is the last character in a full symbol, and
          with a value represented in one byte.
00       The value is 0.
81       The serial number is 1.

```

3.5.3 mmo section mapping

The implementation in BFD uses special data type 80 (decimal) to encapsulate and describe named sections, containing e.g. debug information. If needed, any datum in the encapsulation will be quoted using `lop_quote`. First comes a 32-bit word holding the number of 32-bit words containing the zero-terminated zero-padded segment name. After the name there’s a 32-bit word holding flags describing the section type. Then comes a 64-bit big-endian word with the section length (in bytes), then another with the section start address. Depending on the type of section, the contents might follow, zero-padded to 32-bit boundary. For a loadable section (such as data or code), the contents might follow at some later point, not necessarily immediately, as a `lop_loc` with the same start address as in the section description, followed by the contents. This in effect forms a descriptor that must be emitted before the actual contents. Sections described this way must not overlap.

For areas that don’t have such descriptors, synthetic sections are formed by BFD. Consecutive contents in the two memory areas ‘0x0000...00’ to ‘0x01ff...ff’ and ‘0x2000...00’ to ‘0x20ff...ff’ are entered in sections named `.text` and `.data` respectively. If an area is not otherwise described, but would together with a neighboring lower area be less than ‘0x40000000’ bytes long, it is joined with the lower area and the gap is zero-filled. For other cases, a new section is formed, named `.MMIX.sec.n`. Here, *n* is a number, a running count through the mmo file, starting at 0.

A loadable section specified as:

```

.section secname,"ax"
TETRA 1,2,3,4,-1,-2009
BYTE 80

```

and linked to address ‘0x4’, is represented by the sequence:

```

0x98080050 - lop_spec 80
0x00000002 - two 32-bit words for the section name

```

```

0x7365636e - "secn"
0x616d6500 - "ame\0"
0x00000033 - flags CODE, READONLY, LOAD, ALLOC
0x00000000 - high 32 bits of section length
0x0000001c - section length is 28 bytes; 6 * 4 + 1 + alignment to 32 bits
0x00000000 - high 32 bits of section address
0x00000004 - section address is 4
0x98010002 - 64 bits with address of following data
0x00000000 - high 32 bits of address
0x00000004 - low 32 bits: data starts at address 4
0x00000001 - 1
0x00000002 - 2
0x00000003 - 3
0x00000004 - 4
0xffffffff - -1
0xfffff827 - -2009
0x50000000 - 80 as a byte, padded with zeros.

```

Note that the `lop_spec` wrapping does not include the section contents. Compare this to a non-loaded section specified as:

```

.section thirdsec
TETRA 200001,100002
BYTE 38,40

```

This, when linked to address `'0x200000000000001c'`, is represented by:

```

0x98080050 - lop_spec 80
0x00000002 - two 32-bit words for the section name
0x7365636e - "thir"
0x616d6500 - "dsec"
0x00000010 - flag READONLY
0x00000000 - high 32 bits of section length
0x0000000c - section length is 12 bytes; 2 * 4 + 2 + alignment to 32 bits
0x20000000 - high 32 bits of address
0x0000001c - low 32 bits of address 0x200000000000001c
0x00030d41 - 200001
0x000186a2 - 100002
0x26280000 - 38, 40 as bytes, padded with zeros

```

For the latter example, the section contents must not be loaded in memory, and is therefore specified as part of the special data. The address is usually unimportant but might provide information for e.g. the DWARF 2 debugging format.

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be

added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the

electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail. If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions

of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the GNU Free Documentation License, Version 1.3  
or any later version published by the Free Software Foundation;  
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover  
Texts. A copy of the license is included in the section entitled ‘‘GNU  
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with. . . Texts.” line with this:

```
with the Invariant Sections being list their titles, with  
the Front-Cover Texts being list, and with the Back-Cover Texts  
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

BFD Index

*

*_bfd_generic_make_empty_symbol 45
 *_bfd_new_bfd 166
 *_bfd_new_bfd_contained_in 166
 *bfd_arch_default_fill 166
 *bfd_asprintf 13
 *bfd_create 169
 *bfd_default_reloc_type_lookup 136
 *bfd_demangle 20
 *bfd_elf_bfd_from_remote_memory 168
 *bfd_fdopenr 167
 *bfd_fdopenw 167
 *bfd_follow_build_id_debuglink 171
 *bfd_follow_gnu_debugaltlink 171
 *bfd_follow_gnu_debuglink 170
 *bfd_fopen 166
 *bfd_generic_get_relocated_section_contents 137
 *bfd_get_alt_debug_link_info 170
 *bfd_get_debug_link_info 170
 *bfd_get_linker_section 34
 *bfd_get_next_section_by_name 34
 *bfd_get_relocated_section_contents 20
 *bfd_get_section_by_name 34
 *bfd_get_section_by_name_if 34
 *bfd_get_unique_section_name 34
 *bfd_hash_allocate 183
 *bfd_make_section 35
 *bfd_make_section_anyway 35
 *bfd_make_section_anyway_with_flags 35
 *bfd_make_section_old_way 34
 *bfd_make_section_with_flags 35
 *bfd_mmap 23
 *bfd_openr 166
 *bfd_openr_iovec 168
 *bfd_openr_next_archived_file 47
 *bfd_openstreamr 167
 *bfd_openw 168
 *bfd_reloc_type_lookup 136
 *bfd_sections_find_if 36

-

_bfd_error_handler 13
 _bfd_final_link_relocate 176
 _bfd_free_cached_info 166
 _bfd_generic_link_add_archive_symbols 175
 _bfd_generic_link_add_one_symbol 174
 _bfd_generic_link_check_relocs 178
 _bfd_generic_link_hide_symbol 177, 178
 _bfd_generic_make_empty_symbol 45
 _bfd_generic_set_reloc 137
 _bfd_generic_verify_endian_match 179
 _bfd_get_error_program_name 14

_bfd_link_add_symbols in target vector 173
 _bfd_link_final_link in target vector 175
 _bfd_link_hash_table_create in target vector ... 173
 _bfd_new_bfd 166
 _bfd_new_bfd_contained_in 166
 _bfd_per_xvec_warn 150
 _bfd_relocate_contents 176
 _bfd_section_size_insane 38
 _bfd_set_error_handler_caching 14
 _bfd_stringtab_add 184
 _bfd_stringtab_emit 184
 _bfd_stringtab_free 184
 _bfd_stringtab_init 183
 _bfd_stringtab_size 184
 _bfd_unrecognized_reloc 137
 _bfd_xcoff_stringtab_init 183

A

aout_size_machine_type 187
 aout_size_mkobject 186
 aout_size_mkobject, 187
 aout_size_new_section_hook 187
 aout_size_new_section_hook, 187
 aout_size_set_arch_mach 187
 aout_size_set_arch_mach, 187
 aout_size_some_aout_object_p 186
 aout_size_swap_exec_header_in 186
 aout_size_swap_exec_header_in, 186
 aout_size_swap_exec_header_out 186
 arelent_chain 54

B

bfd_alt_mach_code 20
 bfd_arch_bits_per_address 164
 bfd_arch_bits_per_byte 164
 bfd_arch_default_fill 165
 bfd_arch_get_compatible 163
 bfd_arch_info_type 163, 164, 165
 bfd_arch_list 163
 bfd_arch_mach_octets_per_byte 165
 bfd_architecture 164
 bfd_asprintf 13
 bfd_bread 21
 bfd_bwrite 21, 22
 bfd_cache_close 172
 bfd_cache_close_all 172
 bfd_cache_init 172
 bfd_calc_gnu_debuglink_crc32 170
 bfd_canonicalize_reloc 15
 bfd_canonicalize_syntab 44
 bfd_check_format 47, 48
 bfd_check_format_matches 48
 bfd_check_overflow 54

- bfd_close 169
- bfd_close_all_done 169
- bfd_coff_backend_data 192
- bfd_copy_private_bfd_data 17
- bfd_copy_private_header_data 17
- bfd_copy_private_section_data 37
- bfd_copy_private_symbol_data 45, 46
- bfd_core_file_failing_command 137
- bfd_core_file_failing_signal 137
- bfd_core_file_pid 137
- bfd_create 169
- bfd_create_gnu_debuglink_section 171
- bfd_decode_symclass 45
- bfd_default_arch_struct 164
- bfd_default_compatible 164
- bfd_default_reloc_type_lookup 136
- bfd_default_scan 165
- bfd_default_set_arch_mach 164
- bfd_demangle 20
- bfd_elf_bfd_from_remote_memory 168
- bfd_elf_version_tree** 178
- bfd_emul_get_commonpagesize 20
- bfd_emul_get_maxpagesize 20
- bfd_errmsg 13
- bfd_fdopenr 167
- bfd_fdopenw 167
- bfd_fill_in_gnu_debuglink_section 171
- bfd_find_target 150
- bfd_find_version_for_sym 178
- bfd_flavour_name 151
- bfd_flush 22
- bfd_follow_build_id_debuglink 171
- bfd_follow_gnu_debugaltlink 171
- bfd_follow_gnu_debuglink 170
- bfd_fopen 166
- bfd_format_string 48
- bfd_generic_define_common_symbol 177
- bfd_generic_define_start_stop 178
- bfd_generic_discard_group 38
- bfd_generic_gc_sections 136
- bfd_generic_get_relocated_section_contents 136
- bfd_generic_group_name 38
- bfd_generic_is_group_section 37, 38
- bfd_generic_lookup_section_flags 136
- bfd_generic_merge_sections 136
- bfd_generic_relax_section 136
- bfd_get_alt_debug_link_info 170
- bfd_get_arch 164
- bfd_get_arch_info 165
- bfd_get_arch_size 15
- bfd_get_debug_link_info 170
- bfd_get_error 12
- bfd_get_file_size 23
- bfd_get_gp_size 16
- bfd_get_linker_section 34
- bfd_get_mach 164
- bfd_get_mtime 22
- bfd_get_next_mapent 47
- bfd_get_next_section_by_name 34
- bfd_get_reloc_code_name 136
- bfd_get_reloc_upper_bound 15
- bfd_get_relocated_section_contents 19
- bfd_get_section_by_name 33
- bfd_get_section_by_name_if 34
- bfd_get_section_contents 37
- bfd_get_sign_extend_vma 16
- bfd_get_size 22
- bfd_get_symtab_upper_bound 43
- bfd_get_target_info 150
- bfd_get_unique_section_name 34
- bfd_hash_allocate 179, 183
- bfd_hash_entry** 183
- bfd_hash_insert 183
- bfd_hash_lookup 180, 182
- bfd_hash_newfunc 179, 183
- bfd_hash_rename 183
- bfd_hash_replace 183
- bfd_hash_set_default_size 180, 183
- bfd_hash_table_free 179, 182
- bfd_hash_table_init 179, 182
- bfd_hash_table_init_n 179, 182
- bfd_hash_traverse 180, 183
- bfd_hide_sym_by_version 178
- bfd_init 14
- bfd_install_relocation 55
- bfd_is_local_label 44
- bfd_is_local_label_name 44
- bfd_is_target_special_symbol 44
- bfd_is_undefined_symclass 45
- bfd_iterate_over_targets 151
- bfd_link_check_relocs 178
- bfd_link_hash_entry** 178
- bfd_link_split_section 177
- bfd_lookup_arch 165
- bfd_make_debug_symbol 45
- bfd_make_empty_symbol 45
- bfd_make_readable 169, 170
- bfd_make_section 35
- bfd_make_section_anyway 35
- bfd_make_section_anyway_with_flags 35
- bfd_make_section_old_way 34
- bfd_make_section_with_flags 35
- bfd_make_writable 169
- bfd_malloc_and_get_section 37
- bfd_map_over_sections 36
- bfd_merge_private_bfd_data 178, 179
- bfd_mmap 23
- bfd_octets_per_byte 165
- bfd_open_file 172
- bfd_openr 166
- bfd_openr_iovec 167
- bfd_openr_next_archived_file 47
- bfd_openstreamr 167
- bfd_openw 168
- bfd_perform_relocation 54, 55
- bfd_perror 13

<code>bfd_print_symbol_vandf</code>	44, 45	<code>BFD_ARELOC_BFIN_OR</code>	81
<code>bfd_printable_arch_mach</code>	165	<code>BFD_ARELOC_BFIN_PAGE</code>	82
<code>bfd_printable_name</code>	163	<code>BFD_ARELOC_BFIN_PUSH</code>	81
<code>bfd_record_phdr</code>	20	<code>BFD_ARELOC_BFIN_RSHIFT</code>	81
<code>bfd_reloc_code_real_type</code>	55	<code>BFD_ARELOC_BFIN_SUB</code>	81
<code>bfd_reloc_offset_in_range</code>	54	<code>BFD_ARELOC_BFIN_XOR</code>	81
<code>bfd_reloc_type_lookup</code>	136	<code>BFD_RELOC_12_PCREL</code>	56
<code>bfd_rename_section</code>	36	<code>BFD_RELOC_14</code>	55
<code>bfd_scan_arch</code>	163	<code>BFD_RELOC_16</code>	55
<code>bfd_scan_vma</code>	16	<code>BFD_RELOC_16_BASEREL</code>	57
<code>bfd_section</code>	171	<code>BFD_RELOC_16_GOT_PCREL</code>	56
<code>bfd_section_already_linked</code>	177	<code>BFD_RELOC_16_GOTOFF</code>	56
<code>bfd_section_list_clear</code>	33	<code>BFD_RELOC_16_PCREL</code>	56
<code>bfd_sections_find_if</code>	36	<code>BFD_RELOC_16_PCREL_S2</code>	57
<code>bfd_seek</code>	22	<code>BFD_RELOC_16_PLT_PCREL</code>	56
<code>bfd_set_arch_info</code>	164	<code>BFD_RELOC_16_PLTOFF</code>	56
<code>bfd_set_archive_head</code>	47	<code>BFD_RELOC_16_SECIDX</code>	56
<code>bfd_set_assert_handler</code>	14	<code>BFD_RELOC_23_PCREL_S2</code>	57
<code>bfd_set_default_target</code>	150	<code>BFD_RELOC_24</code>	55
<code>bfd_set_error</code>	12	<code>BFD_RELOC_24_PCREL</code>	56
<code>bfd_set_error_handler</code>	13, 14	<code>BFD_RELOC_24_PLT_PCREL</code>	56
<code>bfd_set_error_program_name</code>	14	<code>BFD_RELOC_26</code>	55
<code>bfd_set_file_flags</code>	15	<code>BFD_RELOC_32</code>	55
<code>bfd_set_filename</code>	172	<code>BFD_RELOC_32_BASEREL</code>	57
<code>bfd_set_format</code>	48	<code>BFD_RELOC_32_GOT_PCREL</code>	56
<code>bfd_set_gp_size</code>	16	<code>BFD_RELOC_32_GOTOFF</code>	56
<code>bfd_set_gp_value</code>	16	<code>BFD_RELOC_32_PCREL</code>	56
<code>bfd_set_input_error</code>	13	<code>BFD_RELOC_32_PCREL_S2</code>	57
<code>bfd_set_private_flags</code>	17	<code>BFD_RELOC_32_PLT_PCREL</code>	56
<code>bfd_set_reloc</code>	15	<code>BFD_RELOC_32_PLTOFF</code>	56
<code>bfd_set_section_contents</code>	36, 37	<code>BFD_RELOC_32_SECREL</code>	56
<code>bfd_set_section_flags</code>	35	<code>BFD_RELOC_386_COPY</code>	67
<code>bfd_set_section_size</code>	36	<code>BFD_RELOC_386_GLOB_DAT</code>	67
<code>bfd_set_start_address</code>	16	<code>BFD_RELOC_386_GOT32</code>	67
<code>bfd_set_symtab</code>	44	<code>BFD_RELOC_386_GOT32X</code>	67
<code>bfd_sprintf_vma</code>	20	<code>BFD_RELOC_386_GOTOFF</code>	67
<code>bfd_stat</code>	22	<code>BFD_RELOC_386_GOTPC</code>	67
<code>bfd_strtab_hash</code>	183, 184	<code>BFD_RELOC_386_IRELATIVE</code>	67
<code>bfd_symbol_info</code>	45	<code>BFD_RELOC_386_JUMP_SLOT</code>	67
<code>bfd_target</code>	150, 151	<code>BFD_RELOC_386_PLT32</code>	67
<code>bfd_target_list</code>	151	<code>BFD_RELOC_386_RELATIVE</code>	67
<code>bfd_tell</code>	22	<code>BFD_RELOC_386_TLS_DESC</code>	67
<code>BFD</code>	1	<code>BFD_RELOC_386_TLS_DESC_CALL</code>	67
<code>BFD canonical format</code>	3	<code>BFD_RELOC_386_TLS_DTPMOD32</code>	67
<code>BFD_ARELOC_BFIN_ADD</code>	81	<code>BFD_RELOC_386_TLS_DTPOFF32</code>	67
<code>BFD_ARELOC_BFIN_ADDR</code>	82	<code>BFD_RELOC_386_TLS_GD</code>	67
<code>BFD_ARELOC_BFIN_AND</code>	81	<code>BFD_RELOC_386_TLS_GOTDESC</code>	67
<code>BFD_ARELOC_BFIN_COMP</code>	82	<code>BFD_RELOC_386_TLS_GOTIE</code>	67
<code>BFD_ARELOC_BFIN_CONST</code>	81	<code>BFD_RELOC_386_TLS_IE</code>	67
<code>BFD_ARELOC_BFIN_DIV</code>	81	<code>BFD_RELOC_386_TLS_IE_32</code>	67
<code>BFD_ARELOC_BFIN_HWPAGE</code>	82	<code>BFD_RELOC_386_TLS_LDM</code>	67
<code>BFD_ARELOC_BFIN_LAND</code>	81	<code>BFD_RELOC_386_TLS_LDO_32</code>	67
<code>BFD_ARELOC_BFIN_LEN</code>	81	<code>BFD_RELOC_386_TLS_LE</code>	67
<code>BFD_ARELOC_BFIN_LOR</code>	81	<code>BFD_RELOC_386_TLS_LE_32</code>	67
<code>BFD_ARELOC_BFIN_LSHIFT</code>	81	<code>BFD_RELOC_386_TLS_TPOFF</code>	67
<code>BFD_ARELOC_BFIN_MOD</code>	81	<code>BFD_RELOC_386_TLS_TPOFF32</code>	67
<code>BFD_ARELOC_BFIN_MULT</code>	81	<code>BFD_RELOC_390_12</code>	99
<code>BFD_ARELOC_BFIN_NEG</code>	81	<code>BFD_RELOC_390_20</code>	102

BFD_RELOC_390_COPY	100	BFD_RELOC_68K_JMP_SLOT	56
BFD_RELOC_390_GLOB_DAT	100	BFD_RELOC_68K_RELATIVE	56
BFD_RELOC_390_GOT12	100	BFD_RELOC_68K_TLS_GD16	56
BFD_RELOC_390_GOT16	100	BFD_RELOC_68K_TLS_GD32	56
BFD_RELOC_390_GOT20	102	BFD_RELOC_68K_TLS_GD8	56
BFD_RELOC_390_GOT64	100	BFD_RELOC_68K_TLS_IE16	57
BFD_RELOC_390_GOTENT	101	BFD_RELOC_68K_TLS_IE32	57
BFD_RELOC_390_GOTOFF64	101	BFD_RELOC_68K_TLS_IE8	57
BFD_RELOC_390_GOTPC	100	BFD_RELOC_68K_TLS_LDM16	56
BFD_RELOC_390_GOTPCDBL	100	BFD_RELOC_68K_TLS_LDM32	56
BFD_RELOC_390_GOTPLT12	101	BFD_RELOC_68K_TLS_LDM8	57
BFD_RELOC_390_GOTPLT16	101	BFD_RELOC_68K_TLS_LDO16	57
BFD_RELOC_390_GOTPLT20	102	BFD_RELOC_68K_TLS_LDO32	57
BFD_RELOC_390_GOTPLT32	101	BFD_RELOC_68K_TLS_LDO8	57
BFD_RELOC_390_GOTPLT64	101	BFD_RELOC_68K_TLS_LE16	57
BFD_RELOC_390_GOTPLTENT	101	BFD_RELOC_68K_TLS_LE32	57
BFD_RELOC_390_IRELATIVE	102	BFD_RELOC_68K_TLS_LE8	57
BFD_RELOC_390_JMP_SLOT	100	BFD_RELOC_8	55
BFD_RELOC_390_PC12DBL	100	BFD_RELOC_8_BASEREL	57
BFD_RELOC_390_PC16DBL	100	BFD_RELOC_8_FFnn	57
BFD_RELOC_390_PC24DBL	100	BFD_RELOC_8_GOT_PCREL	56
BFD_RELOC_390_PC32DBL	100	BFD_RELOC_8_GOTOFF	56
BFD_RELOC_390_PLT12DBL	100	BFD_RELOC_8_PCREL	56
BFD_RELOC_390_PLT16DBL	100	BFD_RELOC_8_PLT_PCREL	56
BFD_RELOC_390_PLT24DBL	100	BFD_RELOC_8_PLTOFF	56
BFD_RELOC_390_PLT32	100	BFD_RELOC_AARCH64_16	119
BFD_RELOC_390_PLT32DBL	100	BFD_RELOC_AARCH64_16_PCREL	119
BFD_RELOC_390_PLT64	101	BFD_RELOC_AARCH64_32	119
BFD_RELOC_390_PLTOFF16	101	BFD_RELOC_AARCH64_32_PCREL	119
BFD_RELOC_390_PLTOFF32	101	BFD_RELOC_AARCH64_64	119
BFD_RELOC_390_PLTOFF64	101	BFD_RELOC_AARCH64_64_PCREL	119
BFD_RELOC_390_RELATIVE	100	BFD_RELOC_AARCH64_ADD_L012	121
BFD_RELOC_390_TLS_DTPMOD	101	BFD_RELOC_AARCH64_ADR_GOT_PAGE	122
BFD_RELOC_390_TLS_DTPOFF	101	BFD_RELOC_AARCH64_ADR_HI21_NC_PCREL	121
BFD_RELOC_390_TLS_GD32	101	BFD_RELOC_AARCH64_ADR_HI21_PCREL	121
BFD_RELOC_390_TLS_GD64	101	BFD_RELOC_AARCH64_ADR_LO21_PCREL	121
BFD_RELOC_390_TLS_GDCALL	101	BFD_RELOC_AARCH64_BRANCH19	121
BFD_RELOC_390_TLS_GOTIE12	101	BFD_RELOC_AARCH64_CALL26	121
BFD_RELOC_390_TLS_GOTIE20	102	BFD_RELOC_AARCH64_COPY	126
BFD_RELOC_390_TLS_GOTIE32	101	BFD_RELOC_AARCH64_GAS_INTERNAL_FIXUP	126
BFD_RELOC_390_TLS_GOTIE64	101	BFD_RELOC_AARCH64_GLOB_DAT	126
BFD_RELOC_390_TLS_IE32	101	BFD_RELOC_AARCH64_GOT_LD_PREL19	122
BFD_RELOC_390_TLS_IE64	101	BFD_RELOC_AARCH64_IRELATIVE	126
BFD_RELOC_390_TLS_IEENT	101	BFD_RELOC_AARCH64_JUMP_SLOT	126
BFD_RELOC_390_TLS_LDCALL	101	BFD_RELOC_AARCH64_JUMP26	121
BFD_RELOC_390_TLS_LDM32	101	BFD_RELOC_AARCH64_LD_GOT_L012_NC	127
BFD_RELOC_390_TLS_LDM64	101	BFD_RELOC_AARCH64_LD_L019_PCREL	120
BFD_RELOC_390_TLS_LD032	101	BFD_RELOC_AARCH64_LD32_GOT_L012_NC	122
BFD_RELOC_390_TLS_LD064	101	BFD_RELOC_AARCH64_LD32_GOTPAGE_L014	122
BFD_RELOC_390_TLS_LE32	101	BFD_RELOC_AARCH64_LD64_GOT_L012_NC	122
BFD_RELOC_390_TLS_LE64	101	BFD_RELOC_AARCH64_LD64_GOTOFF_L015	122
BFD_RELOC_390_TLS_LOAD	101	BFD_RELOC_AARCH64_LD64_GOTPAGE_L015	122
BFD_RELOC_390_TLS_TPOFF	102	BFD_RELOC_AARCH64_LDST_L012	126
BFD_RELOC_64	55	BFD_RELOC_AARCH64_LDST128_L012	121
BFD_RELOC_64_PCREL	56	BFD_RELOC_AARCH64_LDST16_L012	121
BFD_RELOC_64_PLT_PCREL	56	BFD_RELOC_AARCH64_LDST32_L012	121
BFD_RELOC_64_PLTOFF	56	BFD_RELOC_AARCH64_LDST64_L012	121
BFD_RELOC_68K_GLOB_DAT	56	BFD_RELOC_AARCH64_LDST8_L012	121

BFD_RELOC_AARCH64_MOVW_GO	119
BFD_RELOC_AARCH64_MOVW_GO_NC	119
BFD_RELOC_AARCH64_MOVW_GO_S	120
BFD_RELOC_AARCH64_MOVW_G1	119
BFD_RELOC_AARCH64_MOVW_G1_NC	119
BFD_RELOC_AARCH64_MOVW_G1_S	120
BFD_RELOC_AARCH64_MOVW_G2	120
BFD_RELOC_AARCH64_MOVW_G2_NC	120
BFD_RELOC_AARCH64_MOVW_G2_S	120
BFD_RELOC_AARCH64_MOVW_G3	120
BFD_RELOC_AARCH64_MOVW_GOTOFF_GO_NC	122
BFD_RELOC_AARCH64_MOVW_GOTOFF_G1	122
BFD_RELOC_AARCH64_MOVW_PREL_GO	120
BFD_RELOC_AARCH64_MOVW_PREL_GO_NC	120
BFD_RELOC_AARCH64_MOVW_PREL_G1	120
BFD_RELOC_AARCH64_MOVW_PREL_G1_NC	120
BFD_RELOC_AARCH64_MOVW_PREL_G2	120
BFD_RELOC_AARCH64_MOVW_PREL_G2_NC	120
BFD_RELOC_AARCH64_MOVW_PREL_G3	120
BFD_RELOC_AARCH64_NONE	119
BFD_RELOC_AARCH64_NULL	119
BFD_RELOC_AARCH64_RELATIVE	126
BFD_RELOC_AARCH64_RELOC_END	126
BFD_RELOC_AARCH64_RELOC_START	119
BFD_RELOC_AARCH64_TLS_DTPMOD	126
BFD_RELOC_AARCH64_TLS_DTPREL	126
BFD_RELOC_AARCH64_TLS_TPREL	126
BFD_RELOC_AARCH64_TLSDESC	126
BFD_RELOC_AARCH64_TLSDESC_ADD	126
BFD_RELOC_AARCH64_TLSDESC_ADD_LO12	125
BFD_RELOC_AARCH64_TLSDESC_ADR_PAGE21	125
BFD_RELOC_AARCH64_TLSDESC_ADR_PREL21	125
BFD_RELOC_AARCH64_TLSDESC_CALL	126
BFD_RELOC_AARCH64_TLSDESC_LD_LO12_NC	127
BFD_RELOC_AARCH64_TLSDESC_LD_PREL19	125
BFD_RELOC_AARCH64_TLSDESC_LD32_LO12_NC	125
BFD_RELOC_AARCH64_TLSDESC_LD64_LO12	125
BFD_RELOC_AARCH64_TLSDESC_LDR	126
BFD_RELOC_AARCH64_TLSDESC_OFF_GO_NC	126
BFD_RELOC_AARCH64_TLSDESC_OFF_G1	125
BFD_RELOC_AARCH64_TLSD_ADD_LO12_NC	122
BFD_RELOC_AARCH64_TLSD_ADR_PAGE21	122
BFD_RELOC_AARCH64_TLSD_ADR_PREL21	122
BFD_RELOC_AARCH64_TLSD_MOVW_GO_NC	123
BFD_RELOC_AARCH64_TLSD_MOVW_G1	123
BFD_RELOC_AARCH64_TLSIE_ADR_GOTTPREL_PAGE21	123
BFD_RELOC_AARCH64_TLSIE_LD_GOTTPREL_LO12_NC	127
BFD_RELOC_AARCH64_TLSIE_LD_GOTTPREL_PREL19	123
BFD_RELOC_AARCH64_TLSIE_LD32_GOTTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_LD64_GOTTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_MOVW_GOTTPREL_GO_NC	123
BFD_RELOC_AARCH64_TLSIE_MOVW_GOTTPREL_G1	123
BFD_RELOC_AARCH64_TLSIE_ADD_DTPREL_HI12	123
BFD_RELOC_AARCH64_TLSIE_ADD_DTPREL_LO12	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST_DTPREL_LO12	127
BFD_RELOC_AARCH64_TLSIE_ADD_LDST16_DTPREL_LO12	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST16_DTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST16_DTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST32_DTPREL_LO12	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST32_DTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST32_DTPREL_LO12_NC	123
BFD_RELOC_AARCH64_TLSIE_ADD_LDST64_DTPREL_LO12	124
BFD_RELOC_AARCH64_TLSIE_ADD_LDST64_DTPREL_LO12_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_LDST64_DTPREL_LO12_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_LDST8_DTPREL_LO12	124
BFD_RELOC_AARCH64_TLSIE_ADD_LDST8_DTPREL_LO12_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_LDST8_DTPREL_LO12_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_MOVW_DTPREL_GO_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_MOVW_DTPREL_G1	124
BFD_RELOC_AARCH64_TLSIE_ADD_MOVW_DTPREL_G1_NC	124
BFD_RELOC_AARCH64_TLSIE_ADD_MOVW_DTPREL_G2	124
BFD_RELOC_AARCH64_TLSIE_ADD_TPREL_HI12	124
BFD_RELOC_AARCH64_TLSIE_ADD_TPREL_LO12	124
BFD_RELOC_AARCH64_TLSIE_ADD_TPREL_LO12_NC	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST_TPREL_LO12	127
BFD_RELOC_AARCH64_TLSIE_ADD_LDST_TPREL_LO12_NC	127
BFD_RELOC_AARCH64_TLSIE_ADD_LDST16_TPREL_LO12	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST16_TPREL_LO12_NC	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST32_TPREL_LO12	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST32_TPREL_LO12_NC	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST64_TPREL_LO12	125
BFD_RELOC_AARCH64_TLSIE_ADD_LDST64_TPREL_LO12_NC	125

BFD_RELOC_AARCH64_TLSLE_		
LDST8_TPREL_LO12	125	
BFD_RELOC_AARCH64_TLSLE_LDST8_		
TPREL_LO12_NC	125	
BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_GO	124	
BFD_RELOC_AARCH64_TLSLE_		
MOVW_TPREL_GO_NC	124	
BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G1	124	
BFD_RELOC_AARCH64_TLSLE_		
MOVW_TPREL_G1_NC	124	
BFD_RELOC_AARCH64_TLSLE_MOVW_TPREL_G2	124	
BFD_RELOC_AARCH64_TSTBR14	121	
BFD_RELOC_AC_SECTOFF_S9	79	
BFD_RELOC_AC_SECTOFF_S9_1	79	
BFD_RELOC_AC_SECTOFF_S9_2	79	
BFD_RELOC_AC_SECTOFF_U8	79	
BFD_RELOC_AC_SECTOFF_U8_1	79	
BFD_RELOC_AC_SECTOFF_U8_2	79	
BFD_RELOC_ALPHA_BOH	61	
BFD_RELOC_ALPHA_BRSGP	61	
BFD_RELOC_ALPHA_BSR	61	
BFD_RELOC_ALPHA_CODEADDR	61	
BFD_RELOC_ALPHA_DTPMOD64	61	
BFD_RELOC_ALPHA_DTPREL_HI16	61	
BFD_RELOC_ALPHA_DTPREL_LO16	61	
BFD_RELOC_ALPHA_DTPREL16	61	
BFD_RELOC_ALPHA_DTPREL64	61	
BFD_RELOC_ALPHA_ELF_LITERAL	60	
BFD_RELOC_ALPHA_GOTDTPREL16	61	
BFD_RELOC_ALPHA_GOTTREL16	61	
BFD_RELOC_ALPHA_GDISP	60	
BFD_RELOC_ALPHA_GDISP_HI16	60	
BFD_RELOC_ALPHA_GDISP_LO16	60	
BFD_RELOC_ALPHA_GPREL_HI16	61	
BFD_RELOC_ALPHA_GPREL_LO16	61	
BFD_RELOC_ALPHA_HINT	60	
BFD_RELOC_ALPHA_LDA	61	
BFD_RELOC_ALPHA_LINKAGE	61	
BFD_RELOC_ALPHA_LITERAL	60	
BFD_RELOC_ALPHA_LITUSE	60	
BFD_RELOC_ALPHA_NOP	61	
BFD_RELOC_ALPHA_TLSD	61	
BFD_RELOC_ALPHA_TLSDM	61	
BFD_RELOC_ALPHA_TPREL_HI16	61	
BFD_RELOC_ALPHA_TPREL_LO16	61	
BFD_RELOC_ALPHA_TPREL16	61	
BFD_RELOC_ALPHA_TPREL64	61	
BFD_RELOC_ARM_16	78	
BFD_RELOC_ARM_24	78	
BFD_RELOC_ARM_32	78	
BFD_RELOC_ARM_32_ME	79	
BFD_RELOC_ARM_32_ME_S	79	
BFD_RELOC_ARM_32_PCREL	79	
BFD_RELOC_ARM_8	78	
BFD_RELOC_ARM_COPY	79	
BFD_RELOC_ARM_GLOB_DAT	79	
BFD_RELOC_ARM_GOT32	79	
BFD_RELOC_ARM_GOTOFF	79	
BFD_RELOC_ARM_GOTPC	79	
BFD_RELOC_ARM_GOTPC32	79	
BFD_RELOC_ARM_JLI_SECTOFF	79	
BFD_RELOC_ARM_JMP_SLOT	79	
BFD_RELOC_ARM_N16	78	
BFD_RELOC_ARM_N24	78	
BFD_RELOC_ARM_N32	78	
BFD_RELOC_ARM_N32_ME	79	
BFD_RELOC_ARM_N8	78	
BFD_RELOC_ARM_NONE	78	
BFD_RELOC_ARM_NPS_CMEM16	79	
BFD_RELOC_ARM_PC32	79	
BFD_RELOC_ARM_PLT32	79	
BFD_RELOC_ARM_RELATIVE	79	
BFD_RELOC_ARM_S13_PCREL	79	
BFD_RELOC_ARM_S21H_PCREL	78	
BFD_RELOC_ARM_S21H_PCREL_PLT	79	
BFD_RELOC_ARM_S21W_PCREL	78	
BFD_RELOC_ARM_S21W_PCREL_PLT	79	
BFD_RELOC_ARM_S25H_PCREL	78	
BFD_RELOC_ARM_S25H_PCREL_PLT	79	
BFD_RELOC_ARM_S25W_PCREL	78	
BFD_RELOC_ARM_S25W_PCREL_PLT	79	
BFD_RELOC_ARM_SDA	78	
BFD_RELOC_ARM_SDA_12	79	
BFD_RELOC_ARM_SDA_LDST	78	
BFD_RELOC_ARM_SDA_LDST1	78	
BFD_RELOC_ARM_SDA_LDST2	78	
BFD_RELOC_ARM_SDA16_LD	78	
BFD_RELOC_ARM_SDA16_LD1	78	
BFD_RELOC_ARM_SDA16_LD2	78	
BFD_RELOC_ARM_SDA16_ST2	79	
BFD_RELOC_ARM_SDA32	78	
BFD_RELOC_ARM_SDA32_ME	79	
BFD_RELOC_ARM_SECTOFF	78	
BFD_RELOC_ARM_SECTOFF_1	79	
BFD_RELOC_ARM_SECTOFF_2	79	
BFD_RELOC_ARM_SECTOFF_ME	79	
BFD_RELOC_ARM_SECTOFF_ME_1	79	
BFD_RELOC_ARM_SECTOFF_ME_2	79	
BFD_RELOC_ARM_TLS_DTPMOD	79	
BFD_RELOC_ARM_TLS_DTPOFF	79	
BFD_RELOC_ARM_TLS_DTPOFF_S9	79	
BFD_RELOC_ARM_TLS_GD_CALL	79	
BFD_RELOC_ARM_TLS_GD_GOT	79	
BFD_RELOC_ARM_TLS_GD_LD	79	
BFD_RELOC_ARM_TLS_IE_GOT	79	
BFD_RELOC_ARM_TLS_LE_32	79	
BFD_RELOC_ARM_TLS_LE_S9	79	
BFD_RELOC_ARM_TLS_TPOFF	79	
BFD_RELOC_ARM_W	79	
BFD_RELOC_ARM_W_ME	79	
BFD_RELOC_ARM_ADR_IMM	76	
BFD_RELOC_ARM_ADRL_IMMEDIATE	75	
BFD_RELOC_ARM_ALU_PC_GO	74	
BFD_RELOC_ARM_ALU_PC_GO_NC	74	
BFD_RELOC_ARM_ALU_PC_G1	74	
BFD_RELOC_ARM_ALU_PC_G1_NC	74	

BFD_RELOC_ARM_ALU_PC_G2	75	BFD_RELOC_ARM_SBREL32	73
BFD_RELOC_ARM_ALU_SB_G0	75	BFD_RELOC_ARM_SHIFT_IMM	75
BFD_RELOC_ARM_ALU_SB_G0_NC	75	BFD_RELOC_ARM_SMC	75
BFD_RELOC_ARM_ALU_SB_G1	75	BFD_RELOC_ARM_SWI	75
BFD_RELOC_ARM_ALU_SB_G1_NC	75	BFD_RELOC_ARM_T32_ADD_IMM	75
BFD_RELOC_ARM_ALU_SB_G2	75	BFD_RELOC_ARM_T32_ADD_PC12	75
BFD_RELOC_ARM_CP_OFF_IMM	76	BFD_RELOC_ARM_T32_CP_OFF_IMM	76
BFD_RELOC_ARM_CP_OFF_IMM_S2	76	BFD_RELOC_ARM_T32_CP_OFF_IMM_S2	76
BFD_RELOC_ARM_FUNCDESC	74	BFD_RELOC_ARM_T32_IMM12	75
BFD_RELOC_ARM_FUNCDESC_VALUE	74	BFD_RELOC_ARM_T32_IMMEDIATE	75
BFD_RELOC_ARM_GLOB_DAT	74	BFD_RELOC_ARM_T32_OFFSET_IMM	76
BFD_RELOC_ARM_GOT_PREL	74	BFD_RELOC_ARM_T32_OFFSET_U8	76
BFD_RELOC_ARM_GOT32	74	BFD_RELOC_ARM_T32_VLDR_VSTR_OFF_IMM	76
BFD_RELOC_ARM_GOTFUNCDISC	74	BFD_RELOC_ARM_TARGET1	73
BFD_RELOC_ARM_GOTOFF	74	BFD_RELOC_ARM_TARGET2	73
BFD_RELOC_ARM_GOTOFFFUNCDISC	74	BFD_RELOC_ARM_THM_TLS_CALL	74
BFD_RELOC_ARM_GOTPC	74	BFD_RELOC_ARM_THM_TLS_DESCSEQ	74
BFD_RELOC_ARM_HVC	75	BFD_RELOC_ARM_THUMB_ADD	76
BFD_RELOC_ARM_HWLITERAL	76	BFD_RELOC_ARM_THUMB_ALU_ABS_G0_NC	75
BFD_RELOC_ARM_IMMEDIATE	75	BFD_RELOC_ARM_THUMB_ALU_ABS_G1_NC	75
BFD_RELOC_ARM_IN_POOL	76	BFD_RELOC_ARM_THUMB_ALU_ABS_G2_NC	75
BFD_RELOC_ARM_IRELATIVE	75	BFD_RELOC_ARM_THUMB_ALU_ABS_G3_NC	75
BFD_RELOC_ARM_JUMP_SLOT	74	BFD_RELOC_ARM_THUMB_BF13	73
BFD_RELOC_ARM_LDC_PC_G0	75	BFD_RELOC_ARM_THUMB_BF17	73
BFD_RELOC_ARM_LDC_PC_G1	75	BFD_RELOC_ARM_THUMB_BF19	73
BFD_RELOC_ARM_LDC_PC_G2	75	BFD_RELOC_ARM_THUMB_IMM	76
BFD_RELOC_ARM_LDC_SB_G0	75	BFD_RELOC_ARM_THUMB_LOOP12	73
BFD_RELOC_ARM_LDC_SB_G1	75	BFD_RELOC_ARM_THUMB_MOVT	74
BFD_RELOC_ARM_LDC_SB_G2	75	BFD_RELOC_ARM_THUMB_MOVT_PCREL	74
BFD_RELOC_ARM_LDR_IMM	76	BFD_RELOC_ARM_THUMB_MOVW	74
BFD_RELOC_ARM_LDR_PC_G0	75	BFD_RELOC_ARM_THUMB_MOVW_PCREL	74
BFD_RELOC_ARM_LDR_PC_G1	75	BFD_RELOC_ARM_THUMB_OFFSET	73
BFD_RELOC_ARM_LDR_PC_G2	75	BFD_RELOC_ARM_THUMB_SHIFT	76
BFD_RELOC_ARM_LDR_SB_G0	75	BFD_RELOC_ARM_TLS_CALL	74
BFD_RELOC_ARM_LDR_SB_G1	75	BFD_RELOC_ARM_TLS_DESC	74
BFD_RELOC_ARM_LDR_SB_G2	75	BFD_RELOC_ARM_TLS_DESCSEQ	74
BFD_RELOC_ARM_LDRS_PC_G0	75	BFD_RELOC_ARM_TLS_DESCSEQ	74
BFD_RELOC_ARM_LDRS_PC_G1	75	BFD_RELOC_ARM_TLS_DTPMOD32	74
BFD_RELOC_ARM_LDRS_PC_G2	75	BFD_RELOC_ARM_TLS_DTPOFF32	74
BFD_RELOC_ARM_LDRS_SB_G0	75	BFD_RELOC_ARM_TLS_GD32	74
BFD_RELOC_ARM_LDRS_SB_G1	75	BFD_RELOC_ARM_TLS_GD32_FDPIC	74
BFD_RELOC_ARM_LDRS_SB_G2	75	BFD_RELOC_ARM_TLS_GOTDESC	74
BFD_RELOC_ARM_LITERAL	76	BFD_RELOC_ARM_TLS_IE32	74
BFD_RELOC_ARM_MOVT	74	BFD_RELOC_ARM_TLS_IE32_FDPIC	74
BFD_RELOC_ARM_MOVT_PCREL	74	BFD_RELOC_ARM_TLS_LDM32	74
BFD_RELOC_ARM_MOVW	74	BFD_RELOC_ARM_TLS_LDM32_FDPIC	74
BFD_RELOC_ARM_MOVW_PCREL	74	BFD_RELOC_ARM_TLS_LDO32	74
BFD_RELOC_ARM_MULTI	76	BFD_RELOC_ARM_TLS_LE32	74
BFD_RELOC_ARM_OFFSET_IMM	73	BFD_RELOC_ARM_TLS_TPOFF32	74
BFD_RELOC_ARM_OFFSET_IMM8	76	BFD_RELOC_ARM_V4BX	75
BFD_RELOC_ARM_PCREL_BX	72	BFD_RELOC_AVR_13_PCREL	95
BFD_RELOC_ARM_PCREL_BRANCH	72	BFD_RELOC_AVR_16_PM	95
BFD_RELOC_ARM_PCREL_CALL	72	BFD_RELOC_AVR_6	96
BFD_RELOC_ARM_PCREL_JUMP	72	BFD_RELOC_AVR_6_ADIW	96
BFD_RELOC_ARM_PLT32	74	BFD_RELOC_AVR_7_PCREL	95
BFD_RELOC_ARM_PREL31	73	BFD_RELOC_AVR_8_HI	96
BFD_RELOC_ARM_RELATIVE	74	BFD_RELOC_AVR_8_HLO	97
BFD_RELOC_ARM_ROSEGREL32	73	BFD_RELOC_AVR_8_LO	96
		BFD_RELOC_AVR_CALL	96

BFD_RELOC_AVR_DIFF16	97	BFD_RELOC_C6000_EHTYPE	92
BFD_RELOC_AVR_DIFF32	97	BFD_RELOC_C6000_FPHEAD	92
BFD_RELOC_AVR_DIFF8	97	BFD_RELOC_C6000_JUMP_SLOT	92
BFD_RELOC_AVR_HH8_LDI	95	BFD_RELOC_C6000_NOCMP	92
BFD_RELOC_AVR_HH8_LDI_NEG	95	BFD_RELOC_C6000_PCR_H16	92
BFD_RELOC_AVR_HH8_LDI_PM	96	BFD_RELOC_C6000_PCR_L16	92
BFD_RELOC_AVR_HH8_LDI_PM_NEG	96	BFD_RELOC_C6000_PCR_S10	91
BFD_RELOC_AVR_HI8_LDI	95	BFD_RELOC_C6000_PCR_S12	91
BFD_RELOC_AVR_HI8_LDI_GS	96	BFD_RELOC_C6000_PCR_S21	91
BFD_RELOC_AVR_HI8_LDI_NEG	95	BFD_RELOC_C6000_PCR_S7	91
BFD_RELOC_AVR_HI8_LDI_PM	96	BFD_RELOC_C6000_PREL31	91
BFD_RELOC_AVR_HI8_LDI_PM_NEG	96	BFD_RELOC_C6000_SBR_GOT_H16_W	91
BFD_RELOC_AVR_LDI	96	BFD_RELOC_C6000_SBR_GOT_L16_W	91
BFD_RELOC_AVR_LDS_STS_16	97	BFD_RELOC_C6000_SBR_GOT_U15_W	91
BFD_RELOC_AVR_LO8_LDI	95	BFD_RELOC_C6000_SBR_H16_B	91
BFD_RELOC_AVR_LO8_LDI_GS	96	BFD_RELOC_C6000_SBR_H16_H	91
BFD_RELOC_AVR_LO8_LDI_NEG	95	BFD_RELOC_C6000_SBR_H16_W	91
BFD_RELOC_AVR_LO8_LDI_PM	96	BFD_RELOC_C6000_SBR_L16_B	91
BFD_RELOC_AVR_LO8_LDI_PM_NEG	96	BFD_RELOC_C6000_SBR_L16_H	91
BFD_RELOC_AVR_MS8_LDI	95	BFD_RELOC_C6000_SBR_L16_W	91
BFD_RELOC_AVR_MS8_LDI_NEG	95	BFD_RELOC_C6000_SBR_S16	91
BFD_RELOC_AVR_PORT5	97	BFD_RELOC_C6000_SBR_U15_B	91
BFD_RELOC_AVR_PORT6	97	BFD_RELOC_C6000_SBR_U15_H	91
BFD_RELOC_BFIN_10_PCREL	80	BFD_RELOC_C6000_SBR_U15_W	91
BFD_RELOC_BFIN_11_PCREL	80	BFD_RELOC_CKCORE_ADDR_HI16	133
BFD_RELOC_BFIN_12_PCREL_JUMP	80	BFD_RELOC_CKCORE_ADDR_LO16	133
BFD_RELOC_BFIN_12_PCREL_JUMP_S	80	BFD_RELOC_CKCORE_ADDR32	132
BFD_RELOC_BFIN_16_HIGH	80	BFD_RELOC_CKCORE_ADDRGOT	132
BFD_RELOC_BFIN_16_IMM	80	BFD_RELOC_CKCORE_ADDRGOT_HI16	133
BFD_RELOC_BFIN_16_LOW	80	BFD_RELOC_CKCORE_ADDRGOT_LO16	133
BFD_RELOC_BFIN_24_PCREL_CALL_X	80	BFD_RELOC_CKCORE_ADDRPLT	132
BFD_RELOC_BFIN_24_PCREL_JUMP_L	80	BFD_RELOC_CKCORE_ADDRPLT_HI16	133
BFD_RELOC_BFIN_4_PCREL	80	BFD_RELOC_CKCORE_ADDRPLT_LO16	133
BFD_RELOC_BFIN_5_PCREL	80	BFD_RELOC_CKCORE_CALLGRAPH	133
BFD_RELOC_BFIN_FUNCDESC	80	BFD_RELOC_CKCORE_COPY	132
BFD_RELOC_BFIN_FUNCDESC_GOT17M4	80	BFD_RELOC_CKCORE_DOFFSET_IMM18	133
BFD_RELOC_BFIN_FUNCDESC_GOTHI	80	BFD_RELOC_CKCORE_DOFFSET_IMM18BY2	133
BFD_RELOC_BFIN_FUNCDESC_GOTLO	80	BFD_RELOC_CKCORE_DOFFSET_IMM18BY4	133
BFD_RELOC_BFIN_FUNCDESC_GOTOFF17M4	80	BFD_RELOC_CKCORE_DOFFSET_LO16	133
BFD_RELOC_BFIN_FUNCDESC_GOTOFFHI	80	BFD_RELOC_CKCORE_GLOB_DAT	132
BFD_RELOC_BFIN_FUNCDESC_GOTOFFLO	80	BFD_RELOC_CKCORE_GNU_VTENTRY	132
BFD_RELOC_BFIN_FUNCDESC_VALUE	80	BFD_RELOC_CKCORE_GNU_VTINHERIT	132
BFD_RELOC_BFIN_GOT	81	BFD_RELOC_CKCORE_GOT_HI16	133
BFD_RELOC_BFIN_GOT17M4	80	BFD_RELOC_CKCORE_GOT_IMM18BY4	133
BFD_RELOC_BFIN_GOTHI	80	BFD_RELOC_CKCORE_GOT_LO16	133
BFD_RELOC_BFIN_GOTLO	80	BFD_RELOC_CKCORE_GOT12	133
BFD_RELOC_BFIN_GOTOFF17M4	80	BFD_RELOC_CKCORE_GOT32	132
BFD_RELOC_BFIN_GOTOFFHI	80	BFD_RELOC_CKCORE_GOTOFF	132
BFD_RELOC_BFIN_GOTOFFLO	80	BFD_RELOC_CKCORE_GOTOFF_HI16	133
BFD_RELOC_BFIN_PLTPC	81	BFD_RELOC_CKCORE_GOTOFF_IMM18	133
BFD_RELOC_BPF_64	131	BFD_RELOC_CKCORE_GOTOFF_LO16	133
BFD_RELOC_BPF_DISP32	131	BFD_RELOC_CKCORE_GOTPC	132
BFD_RELOC_C6000_ABS_H16	91	BFD_RELOC_CKCORE_GOTPC_HI16	133
BFD_RELOC_C6000_ABS_L16	91	BFD_RELOC_CKCORE_GOTPC_LO16	133
BFD_RELOC_C6000_ABS_S16	91	BFD_RELOC_CKCORE_IRELATIVE	133
BFD_RELOC_C6000_ALIGN	92	BFD_RELOC_CKCORE_JUMP_SLOT	132
BFD_RELOC_C6000_COPY	92	BFD_RELOC_CKCORE_NOJSRI	133
BFD_RELOC_C6000_DSBT_INDEX	91	BFD_RELOC_CKCORE_NONE	132

BFD_RELOC_CKCORE_PCREL_BLOOP_IMM12BY4	133	BFD_RELOC_CR16_REGREL20a	107
BFD_RELOC_CKCORE_PCREL_BLOOP_IMM4BY4	133	BFD_RELOC_CR16_REGREL4	107
BFD_RELOC_CKCORE_PCREL_FLRW_IMM8BY4	133	BFD_RELOC_CR16_REGREL4a	107
BFD_RELOC_CKCORE_PCREL_IMM10BY2	133	BFD_RELOC_CR16_SWITCH16	108
BFD_RELOC_CKCORE_PCREL_IMM10BY4	133	BFD_RELOC_CR16_SWITCH32	108
BFD_RELOC_CKCORE_PCREL_IMM11BY2	132	BFD_RELOC_CR16_SWITCH8	108
BFD_RELOC_CKCORE_PCREL_IMM16BY2	132	BFD_RELOC_CRIS_16_DTPREL	109
BFD_RELOC_CKCORE_PCREL_IMM16BY4	132	BFD_RELOC_CRIS_16_GOT	109
BFD_RELOC_CKCORE_PCREL_IMM18BY2	133	BFD_RELOC_CRIS_16_GOT_GD	109
BFD_RELOC_CKCORE_PCREL_IMM26BY2	132	BFD_RELOC_CRIS_16_GOT_TPREL	109
BFD_RELOC_CKCORE_PCREL_IMM4BY2	132	BFD_RELOC_CRIS_16_GOTPLT	109
BFD_RELOC_CKCORE_PCREL_IMM7BY4	133	BFD_RELOC_CRIS_16_TPREL	109
BFD_RELOC_CKCORE_PCREL_IMM8BY4	132	BFD_RELOC_CRIS_32_DTPREL	109
BFD_RELOC_CKCORE_PCREL_JSR_IMM11BY2	132	BFD_RELOC_CRIS_32_GD	109
BFD_RELOC_CKCORE_PCREL_JSR_IMM26BY2	133	BFD_RELOC_CRIS_32_GOT	109
BFD_RELOC_CKCORE_PCREL32	132	BFD_RELOC_CRIS_32_GOT_GD	109
BFD_RELOC_CKCORE_PLT_HI16	133	BFD_RELOC_CRIS_32_GOT_TPREL	109
BFD_RELOC_CKCORE_PLT_IMM18BY4	133	BFD_RELOC_CRIS_32_GOTPLT	109
BFD_RELOC_CKCORE_PLT_LO16	133	BFD_RELOC_CRIS_32_GOTREL	109
BFD_RELOC_CKCORE_PLT12	133	BFD_RELOC_CRIS_32_IE	109
BFD_RELOC_CKCORE_PLT32	132	BFD_RELOC_CRIS_32_PLT_GOTREL	109
BFD_RELOC_CKCORE_RELATIVE	132	BFD_RELOC_CRIS_32_PLT_PCREL	109
BFD_RELOC_CKCORE_TLS_DTPMOD32	133	BFD_RELOC_CRIS_32_TPREL	109
BFD_RELOC_CKCORE_TLS_DTPOFF32	133	BFD_RELOC_CRIS_BDISP8	108
BFD_RELOC_CKCORE_TLS_GD32	133	BFD_RELOC_CRIS_COPY	108
BFD_RELOC_CKCORE_TLS_IE32	133	BFD_RELOC_CRIS_DTP	109
BFD_RELOC_CKCORE_TLS_LDM32	133	BFD_RELOC_CRIS_DTPMOD	109
BFD_RELOC_CKCORE_TLS_LD032	133	BFD_RELOC_CRIS_GLOB_DAT	108
BFD_RELOC_CKCORE_TLS_LE32	133	BFD_RELOC_CRIS_JUMP_SLOT	108
BFD_RELOC_CKCORE_TLS_TPOFF32	133	BFD_RELOC_CRIS_LAPCQ_OFFSET	108
BFD_RELOC_CKCORE_TOFFSET_LO16	133	BFD_RELOC_CRIS_RELATIVE	108
BFD_RELOC_CR16_ABS20	107	BFD_RELOC_CRIS_SIGNED_16	108
BFD_RELOC_CR16_ABS24	107	BFD_RELOC_CRIS_SIGNED_6	108
BFD_RELOC_CR16_DISP16	107	BFD_RELOC_CRIS_SIGNED_8	108
BFD_RELOC_CR16_DISP20	107	BFD_RELOC_CRIS_UNSIGNED_16	108
BFD_RELOC_CR16_DISP24	107	BFD_RELOC_CRIS_UNSIGNED_4	108
BFD_RELOC_CR16_DISP24a	108	BFD_RELOC_CRIS_UNSIGNED_5	108
BFD_RELOC_CR16_DISP4	107	BFD_RELOC_CRIS_UNSIGNED_6	108
BFD_RELOC_CR16_DISP8	107	BFD_RELOC_CRIS_UNSIGNED_8	108
BFD_RELOC_CR16_GLOB_DAT	108	BFD_RELOC_CRX_ABS16	108
BFD_RELOC_CR16_GOT_REGREL20	108	BFD_RELOC_CRX_ABS32	108
BFD_RELOC_CR16_GOTC_REGREL20	108	BFD_RELOC_CRX_IMM16	108
BFD_RELOC_CR16_IMM16	107	BFD_RELOC_CRX_IMM32	108
BFD_RELOC_CR16_IMM20	107	BFD_RELOC_CRX_NUM16	108
BFD_RELOC_CR16_IMM24	107	BFD_RELOC_CRX_NUM32	108
BFD_RELOC_CR16_IMM32	107	BFD_RELOC_CRX_NUM8	108
BFD_RELOC_CR16_IMM32a	107	BFD_RELOC_CRX_REGREL12	108
BFD_RELOC_CR16_IMM4	107	BFD_RELOC_CRX_REGREL22	108
BFD_RELOC_CR16_IMM8	107	BFD_RELOC_CRX_REGREL28	108
BFD_RELOC_CR16_NUM16	107	BFD_RELOC_CRX_REGREL32	108
BFD_RELOC_CR16_NUM32	107	BFD_RELOC_CRX_REL16	108
BFD_RELOC_CR16_NUM32a	107	BFD_RELOC_CRX_REL24	108
BFD_RELOC_CR16_NUM8	107	BFD_RELOC_CRX_REL32	108
BFD_RELOC_CR16_REGRELO	107	BFD_RELOC_CRX_REL4	108
BFD_RELOC_CR16_REGREL14	107	BFD_RELOC_CRX_REL8	108
BFD_RELOC_CR16_REGREL14a	107	BFD_RELOC_CRX_REL8_CMP	108
BFD_RELOC_CR16_REGREL16	107	BFD_RELOC_CRX_SWITCH16	108
BFD_RELOC_CR16_REGREL20	107	BFD_RELOC_CRX_SWITCH32	108

BFD_RELOC_CRX_SWITCH8	108	BFD_RELOC_FRV_GPREL32	65
BFD_RELOC_CTOR	72	BFD_RELOC_FRV_GPRELHI	65
BFD_RELOC_D10V_10_PCREL_L	82	BFD_RELOC_FRV_GPRELLO	65
BFD_RELOC_D10V_10_PCREL_R	82	BFD_RELOC_FRV_GPRELU12	65
BFD_RELOC_D10V_18	82	BFD_RELOC_FRV_HI16	65
BFD_RELOC_D10V_18_PCREL	82	BFD_RELOC_FRV_LABEL16	65
BFD_RELOC_D30V_15	82	BFD_RELOC_FRV_LABEL24	65
BFD_RELOC_D30V_15_PCREL	82	BFD_RELOC_FRV_LO16	65
BFD_RELOC_D30V_15_PCREL_R	82	BFD_RELOC_FRV_TLSDESC_RELAX	65
BFD_RELOC_D30V_21	82	BFD_RELOC_FRV_TLSDESC_VALUE	65
BFD_RELOC_D30V_21_PCREL	82	BFD_RELOC_FRV_TLSMOFF	65
BFD_RELOC_D30V_21_PCREL_R	83	BFD_RELOC_FRV_TLSMOFF12	65
BFD_RELOC_D30V_32	83	BFD_RELOC_FRV_TLSMOFFHI	65
BFD_RELOC_D30V_32_PCREL	83	BFD_RELOC_FRV_TLSMOFFLO	65
BFD_RELOC_D30V_6	82	BFD_RELOC_FRV_TLSOFF	65
BFD_RELOC_D30V_9_PCREL	82	BFD_RELOC_FRV_TLSOFF_RELAX	65
BFD_RELOC_D30V_9_PCREL_R	82	BFD_RELOC_FT32_10	64
BFD_RELOC_DLX_HI16_S	83	BFD_RELOC_FT32_15	64
BFD_RELOC_DLX_JMP26	83	BFD_RELOC_FT32_17	64
BFD_RELOC_DLX_LO16	83	BFD_RELOC_FT32_18	64
BFD_RELOC_EPIPHANY_HIGH	131	BFD_RELOC_FT32_20	64
BFD_RELOC_EPIPHANY_IMM11	132	BFD_RELOC_FT32_DIFF32	64
BFD_RELOC_EPIPHANY_IMM8	132	BFD_RELOC_FT32_RELAX	64
BFD_RELOC_EPIPHANY_LOW	131	BFD_RELOC_FT32_SCO	64
BFD_RELOC_EPIPHANY_SIMM11	131	BFD_RELOC_FT32_SC1	64
BFD_RELOC_EPIPHANY_SIMM24	131	BFD_RELOC_GPREL16	57
BFD_RELOC_EPIPHANY_SIMM8	131	BFD_RELOC_GPREL32	57
BFD_RELOC_FR30_10_IN_8	92	BFD_RELOC_H8_DIR16A8	110
BFD_RELOC_FR30_12_PCREL	92	BFD_RELOC_H8_DIR16R8	110
BFD_RELOC_FR30_20	92	BFD_RELOC_H8_DIR24A8	110
BFD_RELOC_FR30_48	92	BFD_RELOC_H8_DIR24R8	110
BFD_RELOC_FR30_6_IN_4	92	BFD_RELOC_H8_DIR32A16	110
BFD_RELOC_FR30_8_IN_8	92	BFD_RELOC_H8_DISP32A16	110
BFD_RELOC_FR30_9_IN_8	92	BFD_RELOC_HI16	62
BFD_RELOC_FR30_9_PCREL	92	BFD_RELOC_HI16_BASEREL	57
BFD_RELOC_FRV_FUNCDESC	65	BFD_RELOC_HI16_GOTOFF	56
BFD_RELOC_FRV_FUNCDESC_GOT12	65	BFD_RELOC_HI16_PCREL	62
BFD_RELOC_FRV_FUNCDESC_GOTHI	65	BFD_RELOC_HI16_PLTOFF	56
BFD_RELOC_FRV_FUNCDESC_GOTLO	65	BFD_RELOC_HI16_S	62
BFD_RELOC_FRV_FUNCDESC_GOTOFF12	65	BFD_RELOC_HI16_S_BASEREL	57
BFD_RELOC_FRV_FUNCDESC_GOTOFFHI	65	BFD_RELOC_HI16_S_GOTOFF	56
BFD_RELOC_FRV_FUNCDESC_GOTOFFLO	65	BFD_RELOC_HI16_S_PCREL	62
BFD_RELOC_FRV_FUNCDESC_VALUE	65	BFD_RELOC_HI16_S_PLTOFF	56
BFD_RELOC_FRV_GETTLSOFF	65	BFD_RELOC_HI22	57
BFD_RELOC_FRV_GETTLSOFF_RELAX	65	BFD_RELOC_I370_D12	72
BFD_RELOC_FRV_GOT12	65	BFD_RELOC_IA64_COPY	105
BFD_RELOC_FRV_GOTHI	65	BFD_RELOC_IA64_DIR32LSB	103
BFD_RELOC_FRV_GOTLO	65	BFD_RELOC_IA64_DIR32MSB	103
BFD_RELOC_FRV_GOTOFF12	65	BFD_RELOC_IA64_DIR64LSB	103
BFD_RELOC_FRV_GOTOFFHI	65	BFD_RELOC_IA64_DIR64MSB	103
BFD_RELOC_FRV_GOTOFFLO	65	BFD_RELOC_IA64_DTPMOD64LSB	105
BFD_RELOC_FRV_GOTTLSDESC12	65	BFD_RELOC_IA64_DTPMOD64MSB	105
BFD_RELOC_FRV_GOTTLSDESCHI	65	BFD_RELOC_IA64_DTPREL14	105
BFD_RELOC_FRV_GOTTLSDESCLO	65	BFD_RELOC_IA64_DTPREL22	105
BFD_RELOC_FRV_GOTTLSOFF12	65	BFD_RELOC_IA64_DTPREL32LSB	105
BFD_RELOC_FRV_GOTTLSOFFHI	65	BFD_RELOC_IA64_DTPREL32MSB	105
BFD_RELOC_FRV_GOTTLSOFFLO	65	BFD_RELOC_IA64_DTPREL64I	105
BFD_RELOC_FRV_GPREL12	65	BFD_RELOC_IA64_DTPREL64LSB	105

BFD_RELOC_IA64_DTPREL64MSB	105	BFD_RELOC_IA64_SEGREL32MSB	104
BFD_RELOC_IA64_FPTR32LSB	104	BFD_RELOC_IA64_SEGREL64LSB	104
BFD_RELOC_IA64_FPTR32MSB	104	BFD_RELOC_IA64_SEGREL64MSB	104
BFD_RELOC_IA64_FPTR64I	104	BFD_RELOC_IA64_TPREL14	105
BFD_RELOC_IA64_FPTR64LSB	104	BFD_RELOC_IA64_TPREL22	105
BFD_RELOC_IA64_FPTR64MSB	104	BFD_RELOC_IA64_TPREL64I	105
BFD_RELOC_IA64_GPREL22	103	BFD_RELOC_IA64_TPREL64LSB	105
BFD_RELOC_IA64_GPREL32LSB	104	BFD_RELOC_IA64_TPREL64MSB	105
BFD_RELOC_IA64_GPREL32MSB	103	BFD_RELOC_IP2K_ADDR16CJP	102
BFD_RELOC_IA64_GPREL64I	103	BFD_RELOC_IP2K_BANK	102
BFD_RELOC_IA64_GPREL64LSB	104	BFD_RELOC_IP2K_EX8DATA	103
BFD_RELOC_IA64_GPREL64MSB	104	BFD_RELOC_IP2K_FR_OFFSET	103
BFD_RELOC_IA64_IMM14	103	BFD_RELOC_IP2K_FR9	102
BFD_RELOC_IA64_IMM22	103	BFD_RELOC_IP2K_HI8DATA	103
BFD_RELOC_IA64_IMM64	103	BFD_RELOC_IP2K_HI8INSN	103
BFD_RELOC_IA64_IPLTLSB	105	BFD_RELOC_IP2K_LO8DATA	103
BFD_RELOC_IA64_IPLTMSB	105	BFD_RELOC_IP2K_LO8INSN	103
BFD_RELOC_IA64_LDXMOV	105	BFD_RELOC_IP2K_PAGE3	103
BFD_RELOC_IA64_LTOFF_DTPMOD22	105	BFD_RELOC_IP2K_PC_SKIP	103
BFD_RELOC_IA64_LTOFF_DTPREL22	105	BFD_RELOC_IP2K_TEXT	103
BFD_RELOC_IA64_LTOFF_FPTR22	104	BFD_RELOC_IQ2000_OFFSET_16	113
BFD_RELOC_IA64_LTOFF_FPTR32LSB	104	BFD_RELOC_IQ2000_OFFSET_21	113
BFD_RELOC_IA64_LTOFF_FPTR32MSB	104	BFD_RELOC_IQ2000_UHI16	113
BFD_RELOC_IA64_LTOFF_FPTR64I	104	BFD_RELOC_LARCH_32_PCREL	135
BFD_RELOC_IA64_LTOFF_FPTR64LSB	104	BFD_RELOC_LARCH_64_PCREL	135
BFD_RELOC_IA64_LTOFF_FPTR64MSB	104	BFD_RELOC_LARCH_ABS_HI20	135
BFD_RELOC_IA64_LTOFF_TPREL22	105	BFD_RELOC_LARCH_ABS_LO12	135
BFD_RELOC_IA64_LTOFF22	104	BFD_RELOC_LARCH_ABS64_HI12	135
BFD_RELOC_IA64_LTOFF22X	105	BFD_RELOC_LARCH_ABS64_LO20	135
BFD_RELOC_IA64_LTOFF64I	104	BFD_RELOC_LARCH_ADD_ULEB128	135
BFD_RELOC_IA64_LTV32LSB	104	BFD_RELOC_LARCH_ADD16	134
BFD_RELOC_IA64_LTV32MSB	104	BFD_RELOC_LARCH_ADD24	134
BFD_RELOC_IA64_LTV64LSB	104	BFD_RELOC_LARCH_ADD32	134
BFD_RELOC_IA64_LTV64MSB	104	BFD_RELOC_LARCH_ADD6	135
BFD_RELOC_IA64_PCREL21B	104	BFD_RELOC_LARCH_ADD64	134
BFD_RELOC_IA64_PCREL21BI	104	BFD_RELOC_LARCH_ADD8	134
BFD_RELOC_IA64_PCREL21F	104	BFD_RELOC_LARCH_ALIGN	135
BFD_RELOC_IA64_PCREL21M	104	BFD_RELOC_LARCH_B16	134
BFD_RELOC_IA64_PCREL22	104	BFD_RELOC_LARCH_B21	134
BFD_RELOC_IA64_PCREL32LSB	104	BFD_RELOC_LARCH_B26	135
BFD_RELOC_IA64_PCREL32MSB	104	BFD_RELOC_LARCH_CFA	135
BFD_RELOC_IA64_PCREL60B	104	BFD_RELOC_LARCH_DELETE	135
BFD_RELOC_IA64_PCREL64I	104	BFD_RELOC_LARCH_GOT_HI20	135
BFD_RELOC_IA64_PCREL64LSB	104	BFD_RELOC_LARCH_GOT_LO12	135
BFD_RELOC_IA64_PCREL64MSB	104	BFD_RELOC_LARCH_GOT_PC_HI20	135
BFD_RELOC_IA64_PLTOFF22	104	BFD_RELOC_LARCH_GOT_PC_LO12	135
BFD_RELOC_IA64_PLTOFF64I	104	BFD_RELOC_LARCH_GOT64_HI12	135
BFD_RELOC_IA64_PLTOFF64LSB	104	BFD_RELOC_LARCH_GOT64_LO20	135
BFD_RELOC_IA64_PLTOFF64MSB	104	BFD_RELOC_LARCH_GOT64_PC_HI12	135
BFD_RELOC_IA64_REL32LSB	104	BFD_RELOC_LARCH_GOT64_PC_LO20	135
BFD_RELOC_IA64_REL32MSB	104	BFD_RELOC_LARCH_MARK_LA	134
BFD_RELOC_IA64_REL64LSB	104	BFD_RELOC_LARCH_MARK_PCREL	134
BFD_RELOC_IA64_REL64MSB	104	BFD_RELOC_LARCH_PCALA_HI20	135
BFD_RELOC_IA64_SEGREL32LSB	104	BFD_RELOC_LARCH_PCALA_LO12	135
BFD_RELOC_IA64_SEGREL32MSB	104	BFD_RELOC_LARCH_PCALA64_HI12	135
BFD_RELOC_IA64_SEGREL64LSB	104	BFD_RELOC_LARCH_PCALA64_LO20	135
BFD_RELOC_IA64_SEGREL64MSB	104	BFD_RELOC_LARCH_PCREL20_S2	135
BFD_RELOC_IA64_SEGREL32LSB	104	BFD_RELOC_LARCH_RELAX	135

BFD_RELOC_LARCH_SOP_ADD	134	BFD_RELOC_LM32_CALL	116
BFD_RELOC_LARCH_SOP_AND	134	BFD_RELOC_LM32_COPY	116
BFD_RELOC_LARCH_SOP_ASSERT	134	BFD_RELOC_LM32_GLOB_DAT	116
BFD_RELOC_LARCH_SOP_IF_ELSE	134	BFD_RELOC_LM32_GOTOFF_HI16	116
BFD_RELOC_LARCH_SOP_NOT	134	BFD_RELOC_LM32_GOTOFF_LO16	116
BFD_RELOC_LARCH_SOP_POP_32_S_		BFD_RELOC_LM32_JMP_SLOT	116
0_10_10_16_S2	134	BFD_RELOC_LM32_RELATIVE	116
BFD_RELOC_LARCH_SOP_POP_32_		BFD_RELOC_LO10	57
S_0_5_10_16_S2	134	BFD_RELOC_LO16	62
BFD_RELOC_LARCH_SOP_POP_32_S_10_12	134	BFD_RELOC_LO16_BASEREL	57
BFD_RELOC_LARCH_SOP_POP_32_S_10_16	134	BFD_RELOC_LO16_GOTOFF	56
BFD_RELOC_LARCH_SOP_POP_32_S_10_16_S2	134	BFD_RELOC_LO16_PCREL	62
BFD_RELOC_LARCH_SOP_POP_32_S_10_5	134	BFD_RELOC_LO16_PLTOFF	56
BFD_RELOC_LARCH_SOP_POP_32_S_5_20	134	BFD_RELOC_M32C_HI8	83
BFD_RELOC_LARCH_SOP_POP_32_U	134	BFD_RELOC_M32C_RL_1ADDR	83
BFD_RELOC_LARCH_SOP_POP_32_U_10_12	134	BFD_RELOC_M32C_RL_2ADDR	83
BFD_RELOC_LARCH_SOP_PUSH_ABSOLUTE	134	BFD_RELOC_M32C_RL_JUMP	83
BFD_RELOC_LARCH_SOP_PUSH_DUP	134	BFD_RELOC_M32R_10_PCREL	83
BFD_RELOC_LARCH_SOP_PUSH_GPREL	134	BFD_RELOC_M32R_18_PCREL	83
BFD_RELOC_LARCH_SOP_PUSH_PCREL	134	BFD_RELOC_M32R_24	83
BFD_RELOC_LARCH_SOP_PUSH_PLT_PCREL	134	BFD_RELOC_M32R_26_PCREL	83
BFD_RELOC_LARCH_SOP_PUSH_TLS_GD	134	BFD_RELOC_M32R_26_PLTREL	84
BFD_RELOC_LARCH_SOP_PUSH_TLS_GOT	134	BFD_RELOC_M32R_COPY	84
BFD_RELOC_LARCH_SOP_PUSH_TLS_TPREL	134	BFD_RELOC_M32R_GLOB_DAT	84
BFD_RELOC_LARCH_SOP_SL	134	BFD_RELOC_M32R_GOT16_HI_SLO	84
BFD_RELOC_LARCH_SOP_SR	134	BFD_RELOC_M32R_GOT16_HI_UL0	84
BFD_RELOC_LARCH_SOP_SUB	134	BFD_RELOC_M32R_GOT16_LO	84
BFD_RELOC_LARCH_SUB_ULEB128	135	BFD_RELOC_M32R_GOT24	84
BFD_RELOC_LARCH_SUB16	134	BFD_RELOC_M32R_GOTOFF	84
BFD_RELOC_LARCH_SUB24	134	BFD_RELOC_M32R_GOTOFF_HI_SLO	84
BFD_RELOC_LARCH_SUB32	134	BFD_RELOC_M32R_GOTOFF_HI_UL0	84
BFD_RELOC_LARCH_SUB6	135	BFD_RELOC_M32R_GOTOFF_LO	84
BFD_RELOC_LARCH_SUB64	134	BFD_RELOC_M32R_GOTPC_HI_SLO	84
BFD_RELOC_LARCH_SUB8	134	BFD_RELOC_M32R_GOTPC_HI_UL0	84
BFD_RELOC_LARCH_TLS_DTPMOD32	134	BFD_RELOC_M32R_GOTPC_LO	84
BFD_RELOC_LARCH_TLS_DTPMOD64	134	BFD_RELOC_M32R_GOTPC24	84
BFD_RELOC_LARCH_TLS_DTPREL32	134	BFD_RELOC_M32R_HI16_SLO	83
BFD_RELOC_LARCH_TLS_DTPREL64	134	BFD_RELOC_M32R_HI16_UL0	83
BFD_RELOC_LARCH_TLS_GD_HI20	135	BFD_RELOC_M32R_JMP_SLOT	84
BFD_RELOC_LARCH_TLS_GD_PC_HI20	135	BFD_RELOC_M32R_LO16	83
BFD_RELOC_LARCH_TLS_IE_HI20	135	BFD_RELOC_M32R_RELATIVE	84
BFD_RELOC_LARCH_TLS_IE_LO12	135	BFD_RELOC_M32R_SDA16	83
BFD_RELOC_LARCH_TLS_IE_PC_HI20	135	BFD_RELOC_M68HC11_24	106
BFD_RELOC_LARCH_TLS_IE_PC_LO12	135	BFD_RELOC_M68HC11_3B	105
BFD_RELOC_LARCH_TLS_IE64_HI12	135	BFD_RELOC_M68HC11_HI8	105
BFD_RELOC_LARCH_TLS_IE64_LO20	135	BFD_RELOC_M68HC11_LO16	105
BFD_RELOC_LARCH_TLS_IE64_PC_HI12	135	BFD_RELOC_M68HC11_LO8	105
BFD_RELOC_LARCH_TLS_IE64_PC_LO20	135	BFD_RELOC_M68HC11_PAGE	106
BFD_RELOC_LARCH_TLS_LD_HI20	135	BFD_RELOC_M68HC11_RL_GROUP	105
BFD_RELOC_LARCH_TLS_LD_PC_HI20	135	BFD_RELOC_M68HC11_RL_JUMP	105
BFD_RELOC_LARCH_TLS_LE_HI20	135	BFD_RELOC_M68HC12_10_PCREL	107
BFD_RELOC_LARCH_TLS_LE_LO12	135	BFD_RELOC_M68HC12_16B	107
BFD_RELOC_LARCH_TLS_LE64_HI12	135	BFD_RELOC_M68HC12_5B	106
BFD_RELOC_LARCH_TLS_LE64_LO20	135	BFD_RELOC_M68HC12_9_PCREL	107
BFD_RELOC_LARCH_TLS_TPREL32	134	BFD_RELOC_M68HC12_9B	107
BFD_RELOC_LARCH_TLS_TPREL64	134	BFD_RELOC_M68HC12_HI8XG	107
BFD_RELOC_LM32_16_GOT	116	BFD_RELOC_M68HC12_LO8XG	107
BFD_RELOC_LM32_BRANCH	116	BFD_RELOC_MACH_O_ARM64_ADDEND	117

BFD_RELOC_MACH_O_ARM64_GOT_LOAD_PAGE21...	117	BFD_RELOC_METAG_LOADDR16	93
BFD_RELOC_MACH_O_ARM64_GOT_LOAD_PAGEOFF12	117	BFD_RELOC_METAG_LOOG	93
BFD_RELOC_MACH_O_ARM64_POINTER_TO_GOT	117	BFD_RELOC_METAG_PLT	93
BFD_RELOC_MACH_O_LOCAL_SECTDIFF	116	BFD_RELOC_METAG_REL16	93
BFD_RELOC_MACH_O_PAIR	116	BFD_RELOC_METAG_REL8	93
BFD_RELOC_MACH_O_SECTDIFF	116	BFD_RELOC_METAG_RELATIVE	93
BFD_RELOC_MACH_O_SUBTRACTOR32	116	BFD_RELOC_METAG_RELBRANCH	93
BFD_RELOC_MACH_O_SUBTRACTOR64	116	BFD_RELOC_METAG_RELBRANCH_PLT	93
BFD_RELOC_MACH_O_X86_64_BRANCH32	117	BFD_RELOC_METAG_TLS_DTPMOD	94
BFD_RELOC_MACH_O_X86_64_BRANCH8	117	BFD_RELOC_METAG_TLS_DTPOFF	94
BFD_RELOC_MACH_O_X86_64_GOT	117	BFD_RELOC_METAG_TLS_GD	93
BFD_RELOC_MACH_O_X86_64_GOT_LOAD	117	BFD_RELOC_METAG_TLS_IE	93
BFD_RELOC_MACH_O_X86_64_PCREL32_1	117	BFD_RELOC_METAG_TLS_IENONPIC	93
BFD_RELOC_MACH_O_X86_64_PCREL32_2	117	BFD_RELOC_METAG_TLS_IENONPIC_HI16	93
BFD_RELOC_MACH_O_X86_64_PCREL32_4	117	BFD_RELOC_METAG_TLS_IENONPIC_LO16	94
BFD_RELOC_MACH_O_X86_64_TLV	117	BFD_RELOC_METAG_TLS_LDM	93
BFD_RELOC_MCORE_PCREL_32	92	BFD_RELOC_METAG_TLS_LDO	93
BFD_RELOC_MCORE_PCREL_IMM11BY2	92	BFD_RELOC_METAG_TLS_LDO_HI16	93
BFD_RELOC_MCORE_PCREL_IMM4BY2	92	BFD_RELOC_METAG_TLS_LDO_LO16	93
BFD_RELOC_MCORE_PCREL_IMM8BY4	92	BFD_RELOC_METAG_TLS_LE	94
BFD_RELOC_MCORE_PCREL_JSR_IMM11BY2	92	BFD_RELOC_METAG_TLS_LE_HI16	94
BFD_RELOC_MCORE_RVA	92	BFD_RELOC_METAG_TLS_LE_LO16	94
BFD_RELOC_MEP_16	92	BFD_RELOC_METAG_TLS_TPOFF	94
BFD_RELOC_MEP_32	92	BFD_RELOC_MICROBLAZE_32_GOTOFF	118
BFD_RELOC_MEP_8	92	BFD_RELOC_MICROBLAZE_32_LO	117
BFD_RELOC_MEP_ADDR24A4	93	BFD_RELOC_MICROBLAZE_32_LO_PCREL	117
BFD_RELOC_MEP_GNU_VTENTRY	93	BFD_RELOC_MICROBLAZE_32_ROSDA	117
BFD_RELOC_MEP_GNU_VTINHERIT	93	BFD_RELOC_MICROBLAZE_32_RWSDA	117
BFD_RELOC_MEP_GPREL	93	BFD_RELOC_MICROBLAZE_32_SYM_OP_SYM	117
BFD_RELOC_MEP_HI16S	93	BFD_RELOC_MICROBLAZE_32_TLSDTPMOD	118
BFD_RELOC_MEP_HI16U	93	BFD_RELOC_MICROBLAZE_32_TLSDTPREL	118
BFD_RELOC_MEP_LOW16	93	BFD_RELOC_MICROBLAZE_64_GOT	118
BFD_RELOC_MEP_PCABS24A2	93	BFD_RELOC_MICROBLAZE_64_GOTOFF	118
BFD_RELOC_MEP_PCREL12A2	92	BFD_RELOC_MICROBLAZE_64_GOTPC	118
BFD_RELOC_MEP_PCREL17A2	92	BFD_RELOC_MICROBLAZE_64_NONE	118
BFD_RELOC_MEP_PCREL24A2	93	BFD_RELOC_MICROBLAZE_64_PLT	118
BFD_RELOC_MEP_PCREL8A2	92	BFD_RELOC_MICROBLAZE_64_TEXTPCREL	119
BFD_RELOC_MEP_TPREL	93	BFD_RELOC_MICROBLAZE_64_TEXTREL	119
BFD_RELOC_MEP_TPREL7	93	BFD_RELOC_MICROBLAZE_64_TLS	118
BFD_RELOC_MEP_TPREL7A2	93	BFD_RELOC_MICROBLAZE_64_TLSDTPREL	118
BFD_RELOC_MEP_TPREL7A4	93	BFD_RELOC_MICROBLAZE_64_TLSGD	118
BFD_RELOC_MEP_UIMM24	93	BFD_RELOC_MICROBLAZE_64_TLSGOTTPREL	118
BFD_RELOC_METAG_COPY	93	BFD_RELOC_MICROBLAZE_64_TLSLD	118
BFD_RELOC_METAG_GETSET_GOT	93	BFD_RELOC_MICROBLAZE_64_TLSTPREL	119
BFD_RELOC_METAG_GETSET_GOTOFF	93	BFD_RELOC_MICROBLAZE_COPY	118
BFD_RELOC_METAG_GETSETOFF	93	BFD_RELOC_MICROMIPS_10_PCREL_S1	63
BFD_RELOC_METAG_GLOB_DAT	93	BFD_RELOC_MICROMIPS_16_PCREL_S1	63
BFD_RELOC_METAG_GOTOFF	93	BFD_RELOC_MICROMIPS_7_PCREL_S1	63
BFD_RELOC_METAG_HI16_GOTOFF	93	BFD_RELOC_MICROMIPS_CALL_HI16	63
BFD_RELOC_METAG_HI16_GOTPC	93	BFD_RELOC_MICROMIPS_CALL_LO16	63
BFD_RELOC_METAG_HI16_PLT	93	BFD_RELOC_MICROMIPS_CALL16	63
BFD_RELOC_METAG_HIADDR16	93	BFD_RELOC_MICROMIPS_GOT_DISP	63
BFD_RELOC_METAG_HIOG	93	BFD_RELOC_MICROMIPS_GOT_HI16	63
BFD_RELOC_METAG_JMP_SLOT	93	BFD_RELOC_MICROMIPS_GOT_LO16	63
BFD_RELOC_METAG_LO16_GOTOFF	93	BFD_RELOC_MICROMIPS_GOT_OFST	63
BFD_RELOC_METAG_LO16_GOTPC	93	BFD_RELOC_MICROMIPS_GOT_PAGE	63
BFD_RELOC_METAG_LO16_PLT	93	BFD_RELOC_MICROMIPS_GOT16	63
		BFD_RELOC_MICROMIPS_GPREL16	63

BFD_RELOC_MICROMIPS_HI16	63	BFD_RELOC_MIPS_TLS_TPREL32	64
BFD_RELOC_MICROMIPS_HI16_S	63	BFD_RELOC_MIPS_TLS_TPREL64	64
BFD_RELOC_MICROMIPS_HIGHER	64	BFD_RELOC_MIPS16_16_PCREL_S1	63
BFD_RELOC_MICROMIPS_HIGHEST	64	BFD_RELOC_MIPS16_CALL16	62
BFD_RELOC_MICROMIPS_JALR	64	BFD_RELOC_MIPS16_GOT16	62
BFD_RELOC_MICROMIPS_JMP	61	BFD_RELOC_MIPS16_GPREL	62
BFD_RELOC_MICROMIPS_LITERAL	63	BFD_RELOC_MIPS16_HI16	62
BFD_RELOC_MICROMIPS_LO16	63	BFD_RELOC_MIPS16_HI16_S	62
BFD_RELOC_MICROMIPS_SCN_DISP	64	BFD_RELOC_MIPS16_JMP	62
BFD_RELOC_MICROMIPS_SUB	63	BFD_RELOC_MIPS16_LO16	62
BFD_RELOC_MICROMIPS_TLS_DTPREL_HI16	64	BFD_RELOC_MIPS16_TLS_DTPREL_HI16	62
BFD_RELOC_MICROMIPS_TLS_DTPREL_LO16	64	BFD_RELOC_MIPS16_TLS_DTPREL_LO16	62
BFD_RELOC_MICROMIPS_TLS_GD	64	BFD_RELOC_MIPS16_TLS_GD	62
BFD_RELOC_MICROMIPS_TLS_GOTTPREL	64	BFD_RELOC_MIPS16_TLS_GOTTPREL	62
BFD_RELOC_MICROMIPS_TLS_LDM	64	BFD_RELOC_MIPS16_TLS_LDM	62
BFD_RELOC_MICROMIPS_TLS_TPREL_HI16	64	BFD_RELOC_MIPS16_TLS_TPREL_HI16	62
BFD_RELOC_MICROMIPS_TLS_TPREL_LO16	64	BFD_RELOC_MIPS16_TLS_TPREL_LO16	62
BFD_RELOC_MIPS_16	64	BFD_RELOC_MMIX_ADDR19	94
BFD_RELOC_MIPS_18_PCREL_S3	63	BFD_RELOC_MMIX_ADDR27	94
BFD_RELOC_MIPS_19_PCREL_S2	63	BFD_RELOC_MMIX_BASE_PLUS_OFFSET	95
BFD_RELOC_MIPS_21_PCREL_S2	63	BFD_RELOC_MMIX_CBRANCH	94
BFD_RELOC_MIPS_26_PCREL_S2	63	BFD_RELOC_MMIX_CBRANCH_1	94
BFD_RELOC_MIPS_CALL_HI16	63	BFD_RELOC_MMIX_CBRANCH_2	94
BFD_RELOC_MIPS_CALL_LO16	63	BFD_RELOC_MMIX_CBRANCH_3	94
BFD_RELOC_MIPS_CALL16	63	BFD_RELOC_MMIX_CBRANCH_J	94
BFD_RELOC_MIPS_COPY	64	BFD_RELOC_MMIX_GETA	94
BFD_RELOC_MIPS_DELETE	63	BFD_RELOC_MMIX_GETA_1	94
BFD_RELOC_MIPS_EH	64	BFD_RELOC_MMIX_GETA_2	94
BFD_RELOC_MIPS_GOT_DISP	63	BFD_RELOC_MMIX_GETA_3	94
BFD_RELOC_MIPS_GOT_HI16	63	BFD_RELOC_MMIX_JMP	94
BFD_RELOC_MIPS_GOT_LO16	63	BFD_RELOC_MMIX_JMP_1	94
BFD_RELOC_MIPS_GOT_OFST	63	BFD_RELOC_MMIX_JMP_2	94
BFD_RELOC_MIPS_GOT_PAGE	63	BFD_RELOC_MMIX_JMP_3	94
BFD_RELOC_MIPS_GOT16	63	BFD_RELOC_MMIX_LOCAL	95
BFD_RELOC_MIPS_HIGHER	64	BFD_RELOC_MMIX_PUSHJ	94
BFD_RELOC_MIPS_HIGHEST	63	BFD_RELOC_MMIX_PUSHJ_1	94
BFD_RELOC_MIPS_INSERT_A	63	BFD_RELOC_MMIX_PUSHJ_2	94
BFD_RELOC_MIPS_INSERT_B	63	BFD_RELOC_MMIX_PUSHJ_3	94
BFD_RELOC_MIPS_JALR	64	BFD_RELOC_MMIX_PUSHJ_STUBBABLE	94
BFD_RELOC_MIPS_JMP	61	BFD_RELOC_MMIX_REG	94
BFD_RELOC_MIPS_JUMP_SLOT	64	BFD_RELOC_MMIX_REG_OR_BYTE	94
BFD_RELOC_MIPS_LITERAL	63	BFD_RELOC_MN10300_16_PCREL	66
BFD_RELOC_MIPS_RELGOT	64	BFD_RELOC_MN10300_32_PCREL	66
BFD_RELOC_MIPS_SCN_DISP	64	BFD_RELOC_MN10300_ALIGN	66
BFD_RELOC_MIPS_SHIFT5	63	BFD_RELOC_MN10300_COPY	66
BFD_RELOC_MIPS_SHIFT6	63	BFD_RELOC_MN10300_GLOB_DAT	66
BFD_RELOC_MIPS_SUB	63	BFD_RELOC_MN10300_GOT16	66
BFD_RELOC_MIPS_TLS_DTPMOD32	64	BFD_RELOC_MN10300_GOT24	66
BFD_RELOC_MIPS_TLS_DTPMOD64	64	BFD_RELOC_MN10300_GOT32	66
BFD_RELOC_MIPS_TLS_DTPREL_HI16	64	BFD_RELOC_MN10300_GOTOFF24	65
BFD_RELOC_MIPS_TLS_DTPREL_LO16	64	BFD_RELOC_MN10300_JMP_SLOT	66
BFD_RELOC_MIPS_TLS_DTPREL32	64	BFD_RELOC_MN10300_RELATIVE	66
BFD_RELOC_MIPS_TLS_DTPREL64	64	BFD_RELOC_MN10300_SYM_DIFF	66
BFD_RELOC_MIPS_TLS_GD	64	BFD_RELOC_MN10300_TLS_DTPMOD	66
BFD_RELOC_MIPS_TLS_GOTTPREL	64	BFD_RELOC_MN10300_TLS_DTPOFF	66
BFD_RELOC_MIPS_TLS_LDM	64	BFD_RELOC_MN10300_TLS_GD	66
BFD_RELOC_MIPS_TLS_TPREL_HI16	64	BFD_RELOC_MN10300_TLS_GOTIE	66
BFD_RELOC_MIPS_TLS_TPREL_LO16	64	BFD_RELOC_MN10300_TLS_IE	66

BFD_RELOC_MN10300_TLS_LD	66	BFD_RELOC_NDS32_DWARF2_OP1	86
BFD_RELOC_MN10300_TLS_LDO	66	BFD_RELOC_NDS32_DWARF2_OP2	86
BFD_RELOC_MN10300_TLS_LE	66	BFD_RELOC_NDS32_EMPTY	87
BFD_RELOC_MN10300_TLS_TPOFF	66	BFD_RELOC_NDS32_GLOB_DAT	85
BFD_RELOC_MOXIE_10_PCREL	64	BFD_RELOC_NDS32_GOT_HI20	86
BFD_RELOC_MSP430_10_PCREL	111	BFD_RELOC_NDS32_GOT_L012	86
BFD_RELOC_MSP430_16	111	BFD_RELOC_NDS32_GOT_L015	87
BFD_RELOC_MSP430_16_BYTE	111	BFD_RELOC_NDS32_GOT_L019	87
BFD_RELOC_MSP430_16_PCREL	111	BFD_RELOC_NDS32_GOT_SUFF	87
BFD_RELOC_MSP430_16_PCREL_BYTE	111	BFD_RELOC_NDS32_GOT15S2	87
BFD_RELOC_MSP430_2X_PCREL	111	BFD_RELOC_NDS32_GOT17S2	87
BFD_RELOC_MSP430_ABS_HI16	111	BFD_RELOC_NDS32_GOT20	85
BFD_RELOC_MSP430_ABS8	111	BFD_RELOC_NDS32_GOTOFF	85
BFD_RELOC_MSP430_PREL31	111	BFD_RELOC_NDS32_GOTOFF_HI20	86
BFD_RELOC_MSP430_RL_PCREL	111	BFD_RELOC_NDS32_GOTOFF_L012	86
BFD_RELOC_MSP430_SET_ULEB128	111	BFD_RELOC_NDS32_GOTOFF_L015	87
BFD_RELOC_MSP430_SUB_ULEB128	111	BFD_RELOC_NDS32_GOTOFF_L019	87
BFD_RELOC_MSP430_SYM_DIFF	111	BFD_RELOC_NDS32_GOTOFF_SUFF	87
BFD_RELOC_MSP430X_ABS16	111	BFD_RELOC_NDS32_GOTPC_HI20	86
BFD_RELOC_MSP430X_ABS20_ADR_DST	111	BFD_RELOC_NDS32_GOTPC_L012	86
BFD_RELOC_MSP430X_ABS20_ADR_SRC	111	BFD_RELOC_NDS32_GOTPC20	86
BFD_RELOC_MSP430X_ABS20_EXT_DST	111	BFD_RELOC_NDS32_GOTTPOFF	88
BFD_RELOC_MSP430X_ABS20_EXT_ODST	111	BFD_RELOC_NDS32_GROUP	88
BFD_RELOC_MSP430X_ABS20_EXT_SRC	111	BFD_RELOC_NDS32_HI20	84
BFD_RELOC_MSP430X_PCR16	111	BFD_RELOC_NDS32_INSN16	86
BFD_RELOC_MSP430X_PCR20_CALL	111	BFD_RELOC_NDS32_JMP_SLOT	85
BFD_RELOC_MSP430X_PCR20_EXT_DST	111	BFD_RELOC_NDS32_LABEL	86
BFD_RELOC_MSP430X_PCR20_EXT_ODST	111	BFD_RELOC_NDS32_L012S0	85
BFD_RELOC_MSP430X_PCR20_EXT_SRC	111	BFD_RELOC_NDS32_L012S0_ORI	85
BFD_RELOC_MT_GNU_VTENTRY	111	BFD_RELOC_NDS32_L012S1	85
BFD_RELOC_MT_GNU_VTINHERIT	111	BFD_RELOC_NDS32_L012S2	84
BFD_RELOC_MT_HI16	111	BFD_RELOC_NDS32_L012S2_DP	86
BFD_RELOC_MT_L016	111	BFD_RELOC_NDS32_L012S2_SP	86
BFD_RELOC_MT_PC16	111	BFD_RELOC_NDS32_L012S3	84
BFD_RELOC_MT_PCINSN8	111	BFD_RELOC_NDS32_LOADSTORE	86
BFD_RELOC_NDS32_10_UPCREL	87	BFD_RELOC_NDS32_LONGCALL1	86
BFD_RELOC_NDS32_10IFCU_PCREL	87	BFD_RELOC_NDS32_LONGCALL2	86
BFD_RELOC_NDS32_15_FIXED	86	BFD_RELOC_NDS32_LONGCALL3	86
BFD_RELOC_NDS32_15_PCREL	84	BFD_RELOC_NDS32_LONGCALL4	86
BFD_RELOC_NDS32_17_FIXED	86	BFD_RELOC_NDS32_LONGCALL5	86
BFD_RELOC_NDS32_17_PCREL	84	BFD_RELOC_NDS32_LONGCALL6	86
BFD_RELOC_NDS32_17IFC_PCREL	87	BFD_RELOC_NDS32_LONGJUMP1	86
BFD_RELOC_NDS32_20	84	BFD_RELOC_NDS32_LONGJUMP2	86
BFD_RELOC_NDS32_25_ABS	87	BFD_RELOC_NDS32_LONGJUMP3	86
BFD_RELOC_NDS32_25_FIXED	86	BFD_RELOC_NDS32_LONGJUMP4	86
BFD_RELOC_NDS32_25_PCREL	84	BFD_RELOC_NDS32_LONGJUMP5	86
BFD_RELOC_NDS32_25_PLTREL	85	BFD_RELOC_NDS32_LONGJUMP6	86
BFD_RELOC_NDS32_5	87	BFD_RELOC_NDS32_LONGJUMP7	86
BFD_RELOC_NDS32_9_FIXED	86	BFD_RELOC_NDS32_LSI	88
BFD_RELOC_NDS32_9_PCREL	84	BFD_RELOC_NDS32_MINUEND	87
BFD_RELOC_NDS32_9_PLTREL	85	BFD_RELOC_NDS32_MULCALL_SUFF	87
BFD_RELOC_NDS32_COPY	85	BFD_RELOC_NDS32_PLT_GOT_SUFF	87
BFD_RELOC_NDS32_DATA	87	BFD_RELOC_NDS32_PLT_GOTREL_HI20	86
BFD_RELOC_NDS32_DIFF_ULEB128	87	BFD_RELOC_NDS32_PLT_GOTREL_L012	86
BFD_RELOC_NDS32_DIFF16	87	BFD_RELOC_NDS32_PLT_GOTREL_L015	87
BFD_RELOC_NDS32_DIFF32	87	BFD_RELOC_NDS32_PLT_GOTREL_L019	87
BFD_RELOC_NDS32_DIFF8	87	BFD_RELOC_NDS32_PLT_GOTREL_L020	87
BFD_RELOC_NDS32_DWARF2_LEB	86	BFD_RELOC_NDS32_PLTBLOCK	87

BFD_RELOC_NDS32_PLTREL_HI20	86	BFD_RELOC_NIOS2_CJMP	112
BFD_RELOC_NDS32_PLTREL_LO12	86	BFD_RELOC_NIOS2_COPY	112
BFD_RELOC_NDS32_PTR	87	BFD_RELOC_NIOS2_GLOB_DAT	112
BFD_RELOC_NDS32_PTR_COUNT	87	BFD_RELOC_NIOS2_GOT_HA	112
BFD_RELOC_NDS32_PTR_RESOLVED	87	BFD_RELOC_NIOS2_GOT_LO	112
BFD_RELOC_NDS32_RELATIVE	85	BFD_RELOC_NIOS2_GOT16	112
BFD_RELOC_NDS32_RELAX_ENTRY	87	BFD_RELOC_NIOS2_GOTOFF	112
BFD_RELOC_NDS32_RELAX_REGION_BEGIN	87	BFD_RELOC_NIOS2_GOTOFF_HA	112
BFD_RELOC_NDS32_RELAX_REGION_END	87	BFD_RELOC_NIOS2_GOTOFF_LO	112
BFD_RELOC_NDS32_REMOVE	88	BFD_RELOC_NIOS2_GPREL	112
BFD_RELOC_NDS32_SDA_FP7U2_RELA	87	BFD_RELOC_NIOS2_HI16	112
BFD_RELOC_NDS32_SDA12S2_DP	86	BFD_RELOC_NIOS2_HIADJ16	112
BFD_RELOC_NDS32_SDA12S2_SP	86	BFD_RELOC_NIOS2_IMM5	111
BFD_RELOC_NDS32_SDA15S0	85	BFD_RELOC_NIOS2_IMM6	112
BFD_RELOC_NDS32_SDA15S1	85	BFD_RELOC_NIOS2_IMM8	112
BFD_RELOC_NDS32_SDA15S2	85	BFD_RELOC_NIOS2_JUMP_SLOT	112
BFD_RELOC_NDS32_SDA15S3	85	BFD_RELOC_NIOS2_LO16	112
BFD_RELOC_NDS32_SDA16S3	85	BFD_RELOC_NIOS2_PCREL_HA	112
BFD_RELOC_NDS32_SDA17S2	85	BFD_RELOC_NIOS2_PCREL_LO	112
BFD_RELOC_NDS32_SDA18S1	85	BFD_RELOC_NIOS2_R2_F1I5_2	112
BFD_RELOC_NDS32_SDA19S0	85	BFD_RELOC_NIOS2_R2_I10_1_PCREL	112
BFD_RELOC_NDS32_SUBTRAHEND	87	BFD_RELOC_NIOS2_R2_L5I4X1	112
BFD_RELOC_NDS32_TLS_DESC	88	BFD_RELOC_NIOS2_R2_S12	112
BFD_RELOC_NDS32_TLS_DESC_20	88	BFD_RELOC_NIOS2_R2_T1I7_1_PCREL	112
BFD_RELOC_NDS32_TLS_DESC_ADD	88	BFD_RELOC_NIOS2_R2_T1I7_2	112
BFD_RELOC_NDS32_TLS_DESC_CALL	88	BFD_RELOC_NIOS2_R2_T1X1I6	112
BFD_RELOC_NDS32_TLS_DESC_FUNC	88	BFD_RELOC_NIOS2_R2_T1X1I6_2	113
BFD_RELOC_NDS32_TLS_DESC_HI20	88	BFD_RELOC_NIOS2_R2_T2I4	112
BFD_RELOC_NDS32_TLS_DESC_LO12	88	BFD_RELOC_NIOS2_R2_T2I4_1	112
BFD_RELOC_NDS32_TLS_DESC_MEM	88	BFD_RELOC_NIOS2_R2_T2I4_2	112
BFD_RELOC_NDS32_TLS_DESC_SDA17S2	88	BFD_RELOC_NIOS2_R2_X1I7_2	112
BFD_RELOC_NDS32_TLS_IE_HI20	88	BFD_RELOC_NIOS2_R2_X2L5	112
BFD_RELOC_NDS32_TLS_IE_LO12	88	BFD_RELOC_NIOS2_RELATIVE	112
BFD_RELOC_NDS32_TLS_IE_LO12S2	88	BFD_RELOC_NIOS2_S16	111
BFD_RELOC_NDS32_TLS_IEGP_HI20	88	BFD_RELOC_NIOS2_TLS_DTPMOD	112
BFD_RELOC_NDS32_TLS_IEGP_LO12	88	BFD_RELOC_NIOS2_TLS_DTPREL	112
BFD_RELOC_NDS32_TLS_IEGP_LO12S2	88	BFD_RELOC_NIOS2_TLS_GD16	112
BFD_RELOC_NDS32_TLS_IEGP_LW	88	BFD_RELOC_NIOS2_TLS_IE16	112
BFD_RELOC_NDS32_TLS_LE_15S0	88	BFD_RELOC_NIOS2_TLS_LDM16	112
BFD_RELOC_NDS32_TLS_LE_15S1	88	BFD_RELOC_NIOS2_TLS_LDO16	112
BFD_RELOC_NDS32_TLS_LE_15S2	88	BFD_RELOC_NIOS2_TLS_LE16	112
BFD_RELOC_NDS32_TLS_LE_20	88	BFD_RELOC_NIOS2_TLS_TPREL	112
BFD_RELOC_NDS32_TLS_LE_ADD	88	BFD_RELOC_NIOS2_U16	111
BFD_RELOC_NDS32_TLS_LE_HI20	88	BFD_RELOC_NIOS2_UJMP	112
BFD_RELOC_NDS32_TLS_LE_LO12	88	BFD_RELOC_NONE	57
BFD_RELOC_NDS32_TLS_LE_LS	88	BFD_RELOC_NS32K_DISP_16	68
BFD_RELOC_NDS32_TPOFF	88	BFD_RELOC_NS32K_DISP_16_PCREL	68
BFD_RELOC_NDS32_TRAN	87	BFD_RELOC_NS32K_DISP_32	68
BFD_RELOC_NDS32_UPDATE_TA	87	BFD_RELOC_NS32K_DISP_32_PCREL	68
BFD_RELOC_NDS32_WORD_9_PCREL	84	BFD_RELOC_NS32K_DISP_8	68
BFD_RELOC_NIOS2_ALIGN	112	BFD_RELOC_NS32K_DISP_8_PCREL	68
BFD_RELOC_NIOS2_CACHE_OPX	112	BFD_RELOC_NS32K_IMM_16	68
BFD_RELOC_NIOS2_CALL_HA	112	BFD_RELOC_NS32K_IMM_16_PCREL	68
BFD_RELOC_NIOS2_CALL_LO	112	BFD_RELOC_NS32K_IMM_32	68
BFD_RELOC_NIOS2_CALL16	112	BFD_RELOC_NS32K_IMM_32_PCREL	68
BFD_RELOC_NIOS2_CALL26	111	BFD_RELOC_NS32K_IMM_8	68
BFD_RELOC_NIOS2_CALL26_NOAT	112	BFD_RELOC_NS32K_IMM_8_PCREL	68
BFD_RELOC_NIOS2_CALLR	112	BFD_RELOC_OR1K_COPY	110

BFD_RELOC_OR1K_GLOB_DAT	110	BFD_RELOC_PPC_DTPREL	71
BFD_RELOC_OR1K_GOT_AHI16	109	BFD_RELOC_PPC_DTPREL16	71
BFD_RELOC_OR1K_GOT_LO13	109	BFD_RELOC_PPC_DTPREL16_HA	71
BFD_RELOC_OR1K_GOT_PG21	109	BFD_RELOC_PPC_DTPREL16_HI	71
BFD_RELOC_OR1K_GOT16	109	BFD_RELOC_PPC_DTPREL16_LO	71
BFD_RELOC_OR1K_GOTOFF_SLO16	110	BFD_RELOC_PPC_EMB_BIT_FLD	69
BFD_RELOC_OR1K_GOTPC_HI16	109	BFD_RELOC_PPC_EMB_MRKREF	69
BFD_RELOC_OR1K_GOTPC_LO16	109	BFD_RELOC_PPC_EMB_NADDR16	69
BFD_RELOC_OR1K_JMP_SLOT	110	BFD_RELOC_PPC_EMB_NADDR16_HA	69
BFD_RELOC_OR1K_LO13	109	BFD_RELOC_PPC_EMB_NADDR16_HI	69
BFD_RELOC_OR1K_PCREL_PG21	109	BFD_RELOC_PPC_EMB_NADDR16_LO	69
BFD_RELOC_OR1K_PLT26	109	BFD_RELOC_PPC_EMB_NADDR32	69
BFD_RELOC_OR1K_PLTA26	109	BFD_RELOC_PPC_EMB_RELSDA	69
BFD_RELOC_OR1K_REL_26	109	BFD_RELOC_PPC_EMB_RELSEC16	69
BFD_RELOC_OR1K_RELATIVE	110	BFD_RELOC_PPC_EMB_RELST_HA	69
BFD_RELOC_OR1K_SLO13	109	BFD_RELOC_PPC_EMB_RELST_HI	69
BFD_RELOC_OR1K_SLO16	109	BFD_RELOC_PPC_EMB_RELST_LO	69
BFD_RELOC_OR1K_TLS_DTPMOD	110	BFD_RELOC_PPC_EMB_SDA21	69
BFD_RELOC_OR1K_TLS_DTPOFF	110	BFD_RELOC_PPC_EMB_SDA2I16	69
BFD_RELOC_OR1K_TLS_GD_HI16	110	BFD_RELOC_PPC_EMB_SDA2REL	69
BFD_RELOC_OR1K_TLS_GD_LO13	110	BFD_RELOC_PPC_EMB_SDAI16	69
BFD_RELOC_OR1K_TLS_GD_LO16	110	BFD_RELOC_PPC_GLOB_DAT	69
BFD_RELOC_OR1K_TLS_GD_PG21	110	BFD_RELOC_PPC_GOT_DTPREL16	71
BFD_RELOC_OR1K_TLS_IE_AHI16	110	BFD_RELOC_PPC_GOT_DTPREL16_HA	71
BFD_RELOC_OR1K_TLS_IE_HI16	110	BFD_RELOC_PPC_GOT_DTPREL16_HI	71
BFD_RELOC_OR1K_TLS_IE_LO13	110	BFD_RELOC_PPC_GOT_DTPREL16_LO	71
BFD_RELOC_OR1K_TLS_IE_LO16	110	BFD_RELOC_PPC_GOT_TLSGD16	71
BFD_RELOC_OR1K_TLS_IE_PG21	110	BFD_RELOC_PPC_GOT_TLSGD16_HA	71
BFD_RELOC_OR1K_TLS_LDM_HI16	110	BFD_RELOC_PPC_GOT_TLSGD16_HI	71
BFD_RELOC_OR1K_TLS_LDM_LO13	110	BFD_RELOC_PPC_GOT_TLSGD16_LO	71
BFD_RELOC_OR1K_TLS_LDM_LO16	110	BFD_RELOC_PPC_GOT_TLSLD16	71
BFD_RELOC_OR1K_TLS_LDM_PG21	110	BFD_RELOC_PPC_GOT_TLSLD16_HA	71
BFD_RELOC_OR1K_TLS_LDO_HI16	110	BFD_RELOC_PPC_GOT_TLSLD16_HI	71
BFD_RELOC_OR1K_TLS_LDO_LO16	110	BFD_RELOC_PPC_GOT_TLSLD16_LO	71
BFD_RELOC_OR1K_TLS_LE_AHI16	110	BFD_RELOC_PPC_GOT_TPREL16	71
BFD_RELOC_OR1K_TLS_LE_HI16	110	BFD_RELOC_PPC_GOT_TPREL16_HA	71
BFD_RELOC_OR1K_TLS_LE_LO16	110	BFD_RELOC_PPC_GOT_TPREL16_HI	71
BFD_RELOC_OR1K_TLS_LE_SLO16	110	BFD_RELOC_PPC_GOT_TPREL16_LO	71
BFD_RELOC_OR1K_TLS_TPOFF	110	BFD_RELOC_PPC_JMP_SLOT	69
BFD_RELOC_PDP11_DISP_6_PCREL	68	BFD_RELOC_PPC_LOCAL24PC	69
BFD_RELOC_PDP11_DISP_8_PCREL	68	BFD_RELOC_PPC_NEG	69
BFD_RELOC_PJ_CODE_DIR16	68	BFD_RELOC_PPC_REL16DX_HA	69
BFD_RELOC_PJ_CODE_DIR32	68	BFD_RELOC_PPC_RELATIVE	69
BFD_RELOC_PJ_CODE_HI16	68	BFD_RELOC_PPC_TLS	71
BFD_RELOC_PJ_CODE_LO16	68	BFD_RELOC_PPC_TLSGD	71
BFD_RELOC_PJ_CODE_REL16	68	BFD_RELOC_PPC_TLSIE	71
BFD_RELOC_PJ_CODE_REL32	68	BFD_RELOC_PPC_TLSLD	71
BFD_RELOC_PPC_16DX_HA	69	BFD_RELOC_PPC_TLSLE	71
BFD_RELOC_PPC_B16	68	BFD_RELOC_PPC_TLSM	71
BFD_RELOC_PPC_B16_BRNTAKEN	68	BFD_RELOC_PPC_TLSML	71
BFD_RELOC_PPC_B16_BRTAKEN	68	BFD_RELOC_PPC_TOC16	68
BFD_RELOC_PPC_B26	68	BFD_RELOC_PPC_TOC16_HI	68
BFD_RELOC_PPC_BA16	68	BFD_RELOC_PPC_TOC16_LO	68
BFD_RELOC_PPC_BA16_BRNTAKEN	69	BFD_RELOC_PPC_TPREL	71
BFD_RELOC_PPC_BA16_BRTAKEN	68	BFD_RELOC_PPC_TPREL16	71
BFD_RELOC_PPC_BA26	68	BFD_RELOC_PPC_TPREL16_HA	71
BFD_RELOC_PPC_COPY	69	BFD_RELOC_PPC_TPREL16_HI	71
BFD_RELOC_PPC_DTPMOD	71	BFD_RELOC_PPC_TPREL16_LO	71

BFD_RELOC_PPC_VLE_HA16A	69	BFD_RELOC_PPC64_PLTGOT16_HA	70
BFD_RELOC_PPC_VLE_HA16D	69	BFD_RELOC_PPC64_PLTGOT16_HI	70
BFD_RELOC_PPC_VLE_HI16A	69	BFD_RELOC_PPC64_PLTGOT16_LO	70
BFD_RELOC_PPC_VLE_HI16D	69	BFD_RELOC_PPC64_PLTGOT16_LO_DS	70
BFD_RELOC_PPC_VLE_LO16A	69	BFD_RELOC_PPC64_REL16_HIGH	70
BFD_RELOC_PPC_VLE_LO16D	69	BFD_RELOC_PPC64_REL16_HIGHA	70
BFD_RELOC_PPC_VLE_REL15	69	BFD_RELOC_PPC64_REL16_HIGHER	70
BFD_RELOC_PPC_VLE_REL24	69	BFD_RELOC_PPC64_REL16_HIGHER34	70
BFD_RELOC_PPC_VLE_REL8	69	BFD_RELOC_PPC64_REL16_HIGHERA	70
BFD_RELOC_PPC_VLE_SDA21	69	BFD_RELOC_PPC64_REL16_HIGHERA34	70
BFD_RELOC_PPC_VLE_SDA21_LO	69	BFD_RELOC_PPC64_REL16_HIGHEST	70
BFD_RELOC_PPC_VLE_SDAREL_HA16A	69	BFD_RELOC_PPC64_REL16_HIGHEST34	70
BFD_RELOC_PPC_VLE_SDAREL_HA16D	69	BFD_RELOC_PPC64_REL16_HIGHESTA	70
BFD_RELOC_PPC_VLE_SDAREL_HI16A	69	BFD_RELOC_PPC64_REL16_HIGHESTA34	70
BFD_RELOC_PPC_VLE_SDAREL_HI16D	69	BFD_RELOC_PPC64_REL24_NOTOC	70
BFD_RELOC_PPC_VLE_SDAREL_LO16A	69	BFD_RELOC_PPC64_REL24_P9NOTOC	70
BFD_RELOC_PPC_VLE_SDAREL_LO16D	69	BFD_RELOC_PPC64_SECTOFF_DS	70
BFD_RELOC_PPC64_ADDR16_DS	70	BFD_RELOC_PPC64_SECTOFF_LO_DS	70
BFD_RELOC_PPC64_ADDR16_HIGH	70	BFD_RELOC_PPC64_TLS_PCREL	72
BFD_RELOC_PPC64_ADDR16_HIGHA	70	BFD_RELOC_PPC64_TLSGD	71
BFD_RELOC_PPC64_ADDR16_HIGHER34	70	BFD_RELOC_PPC64_TLSIE	71
BFD_RELOC_PPC64_ADDR16_HIGHERA34	70	BFD_RELOC_PPC64_TLSLD	71
BFD_RELOC_PPC64_ADDR16_HIGHEST34	70	BFD_RELOC_PPC64_TLSLE	71
BFD_RELOC_PPC64_ADDR16_HIGHESTA34	70	BFD_RELOC_PPC64_TLSM	71
BFD_RELOC_PPC64_ADDR16_LO_DS	70	BFD_RELOC_PPC64_TLSML	71
BFD_RELOC_PPC64_ADDR16_LOCAL	70	BFD_RELOC_PPC64_TOC	70
BFD_RELOC_PPC64_D28	70	BFD_RELOC_PPC64_TOC16_DS	70
BFD_RELOC_PPC64_D34	70	BFD_RELOC_PPC64_TOC16_HA	70
BFD_RELOC_PPC64_D34_HA30	70	BFD_RELOC_PPC64_TOC16_HI	70
BFD_RELOC_PPC64_D34_HI30	70	BFD_RELOC_PPC64_TOC16_LO	69
BFD_RELOC_PPC64_D34_LO	70	BFD_RELOC_PPC64_TOC16_LO_DS	70
BFD_RELOC_PPC64_DTPREL16_DS	72	BFD_RELOC_PPC64_TPREL16_DS	71
BFD_RELOC_PPC64_DTPREL16_HIGH	72	BFD_RELOC_PPC64_TPREL16_HIGH	71
BFD_RELOC_PPC64_DTPREL16_HIGHA	72	BFD_RELOC_PPC64_TPREL16_HIGHA	71
BFD_RELOC_PPC64_DTPREL16_HIGHER	72	BFD_RELOC_PPC64_TPREL16_HIGHER	71
BFD_RELOC_PPC64_DTPREL16_HIGHERA	72	BFD_RELOC_PPC64_TPREL16_HIGHERA	72
BFD_RELOC_PPC64_DTPREL16_HIGHEST	72	BFD_RELOC_PPC64_TPREL16_HIGHEST	72
BFD_RELOC_PPC64_DTPREL16_HIGHESTA	72	BFD_RELOC_PPC64_TPREL16_HIGHESTA	72
BFD_RELOC_PPC64_DTPREL16_LO_DS	72	BFD_RELOC_PPC64_TPREL16_LO_DS	71
BFD_RELOC_PPC64_DTPREL34	72	BFD_RELOC_PPC64_TPREL34	72
BFD_RELOC_PPC64_ENTRY	70	BFD_RELOC_PRU_16_PMEM	113
BFD_RELOC_PPC64_GOT_DTPREL_PCREL34	72	BFD_RELOC_PRU_32_PMEM	113
BFD_RELOC_PPC64_GOT_PCREL34	70	BFD_RELOC_PRU_GNU_DIFF16	113
BFD_RELOC_PPC64_GOT_TLSGD_PCREL34	72	BFD_RELOC_PRU_GNU_DIFF16_PMEM	113
BFD_RELOC_PPC64_GOT_TLSLD_PCREL34	72	BFD_RELOC_PRU_GNU_DIFF32	113
BFD_RELOC_PPC64_GOT_TPREL_PCREL34	72	BFD_RELOC_PRU_GNU_DIFF32_PMEM	113
BFD_RELOC_PPC64_GOT16_DS	70	BFD_RELOC_PRU_GNU_DIFF8	113
BFD_RELOC_PPC64_GOT16_LO_DS	70	BFD_RELOC_PRU_LDI32	113
BFD_RELOC_PPC64_HIGHER	69	BFD_RELOC_PRU_S10_PCREL	113
BFD_RELOC_PPC64_HIGHER_S	69	BFD_RELOC_PRU_U16	113
BFD_RELOC_PPC64_HIGHEST	69	BFD_RELOC_PRU_U16_PMEMIMM	113
BFD_RELOC_PPC64_HIGHEST_S	69	BFD_RELOC_PRU_U8_PCREL	113
BFD_RELOC_PPC64_PCREL28	70	BFD_RELOC_RELC	110
BFD_RELOC_PPC64_PCREL34	70	BFD_RELOC_RISCV_32_PCREL	98
BFD_RELOC_PPC64_PLT_PCREL34	70	BFD_RELOC_RISCV_ADD16	97
BFD_RELOC_PPC64_PLT16_LO_DS	70	BFD_RELOC_RISCV_ADD32	97
BFD_RELOC_PPC64_PLTGOT16	70	BFD_RELOC_RISCV_ADD64	97
BFD_RELOC_PPC64_PLTGOT16_DS	70	BFD_RELOC_RISCV_ADD8	97

BFD_RELOC_RISCV_ALIGN	98	BFD_RELOC_RL78_ABS8	98
BFD_RELOC_RISCV_CALL	97	BFD_RELOC_RL78_CODE	99
BFD_RELOC_RISCV_CALL_PLT	97	BFD_RELOC_RL78_DIFF	98
BFD_RELOC_RISCV_CFA	98	BFD_RELOC_RL78_DIR3U_PCREL	98
BFD_RELOC_RISCV_GOT_HI20	97	BFD_RELOC_RL78_GPRELB	98
BFD_RELOC_RISCV_GPREL_I	98	BFD_RELOC_RL78_GPRELL	98
BFD_RELOC_RISCV_GPREL_S	98	BFD_RELOC_RL78_GPRELW	98
BFD_RELOC_RISCV_GPREL12_I	97	BFD_RELOC_RL78_HI16	99
BFD_RELOC_RISCV_GPREL12_S	97	BFD_RELOC_RL78_HI8	99
BFD_RELOC_RISCV_HI20	97	BFD_RELOC_RL78_LO16	99
BFD_RELOC_RISCV_JMP	97	BFD_RELOC_RL78_NEG16	98
BFD_RELOC_RISCV_LO12_I	97	BFD_RELOC_RL78_NEG24	98
BFD_RELOC_RISCV_LO12_S	97	BFD_RELOC_RL78_NEG32	98
BFD_RELOC_RISCV_PCREL_HI20	97	BFD_RELOC_RL78_NEG8	98
BFD_RELOC_RISCV_PCREL_LO12_I	97	BFD_RELOC_RL78_OP_AND	98
BFD_RELOC_RISCV_PCREL_LO12_S	97	BFD_RELOC_RL78_OP_NEG	98
BFD_RELOC_RISCV_RELAX	98	BFD_RELOC_RL78_OP_SHRA	98
BFD_RELOC_RISCV_RVC_BRANCH	98	BFD_RELOC_RL78_OP_SUBTRACT	98
BFD_RELOC_RISCV_RVC_JUMP	98	BFD_RELOC_RL78_RELAX	99
BFD_RELOC_RISCV_RVC_LUI	98	BFD_RELOC_RL78_SADDR	99
BFD_RELOC_RISCV_SET_ULEB128	98	BFD_RELOC_RL78_SYM	98
BFD_RELOC_RISCV_SET16	98	BFD_RELOC_RVA	57
BFD_RELOC_RISCV_SET32	98	BFD_RELOC_RX_16_OP	99
BFD_RELOC_RISCV_SET6	98	BFD_RELOC_RX_16U	99
BFD_RELOC_RISCV_SET8	98	BFD_RELOC_RX_24_OP	99
BFD_RELOC_RISCV_SUB_ULEB128	98	BFD_RELOC_RX_24U	99
BFD_RELOC_RISCV_SUB16	97	BFD_RELOC_RX_32_OP	99
BFD_RELOC_RISCV_SUB32	97	BFD_RELOC_RX_8U	99
BFD_RELOC_RISCV_SUB6	98	BFD_RELOC_RX_ABS16	99
BFD_RELOC_RISCV_SUB64	97	BFD_RELOC_RX_ABS16_REV	99
BFD_RELOC_RISCV_SUB8	97	BFD_RELOC_RX_ABS16U	99
BFD_RELOC_RISCV_TLS_DTPMOD32	97	BFD_RELOC_RX_ABS16UL	99
BFD_RELOC_RISCV_TLS_DTPMOD64	98	BFD_RELOC_RX_ABS16UW	99
BFD_RELOC_RISCV_TLS_DTPREL32	98	BFD_RELOC_RX_ABS32	99
BFD_RELOC_RISCV_TLS_DTPREL64	98	BFD_RELOC_RX_ABS32_REV	99
BFD_RELOC_RISCV_TLS_GD_HI20	97	BFD_RELOC_RX_ABS8	99
BFD_RELOC_RISCV_TLS_GOT_HI20	97	BFD_RELOC_RX_DIFF	99
BFD_RELOC_RISCV_TLS_TPREL32	98	BFD_RELOC_RX_DIR3U_PCREL	99
BFD_RELOC_RISCV_TLS_TPREL64	98	BFD_RELOC_RX_GPRELB	99
BFD_RELOC_RISCV_TPREL_ADD	97	BFD_RELOC_RX_GPRELL	99
BFD_RELOC_RISCV_TPREL_HI20	97	BFD_RELOC_RX_GPRELW	99
BFD_RELOC_RISCV_TPREL_I	98	BFD_RELOC_RX_NEG16	99
BFD_RELOC_RISCV_TPREL_LO12_I	97	BFD_RELOC_RX_NEG24	99
BFD_RELOC_RISCV_TPREL_LO12_S	97	BFD_RELOC_RX_NEG32	99
BFD_RELOC_RISCV_TPREL_S	98	BFD_RELOC_RX_NEG8	99
BFD_RELOC_RL78_16_OP	98	BFD_RELOC_RX_OP_NEG	99
BFD_RELOC_RL78_16U	98	BFD_RELOC_RX_OP_SUBTRACT	99
BFD_RELOC_RL78_24_OP	98	BFD_RELOC_RX_RELAX	99
BFD_RELOC_RL78_24U	98	BFD_RELOC_RX_SYM	99
BFD_RELOC_RL78_32_OP	98	BFD_RELOC_S12Z_15_PCREL	107
BFD_RELOC_RL78_8U	98	BFD_RELOC_S12Z_OPR	134
BFD_RELOC_RL78_ABS16	98	BFD_RELOC_SCORE_BCMP	102
BFD_RELOC_RL78_ABS16_REV	99	BFD_RELOC_SCORE_BRANCH	102
BFD_RELOC_RL78_ABS16U	99	BFD_RELOC_SCORE_CALL15	102
BFD_RELOC_RL78_ABS16UL	99	BFD_RELOC_SCORE_DUMMY_HI16	102
BFD_RELOC_RL78_ABS16UW	99	BFD_RELOC_SCORE_DUMMY2	102
BFD_RELOC_RL78_ABS32	99	BFD_RELOC_SCORE_GOT_LO16	102
BFD_RELOC_RL78_ABS32_REV	99	BFD_RELOC_SCORE_GOT15	102

BFD_RELOC_SCORE_GPREL15	102	BFD_RELOC_SH_IMM3U	76
BFD_RELOC_SCORE_IMM30	102	BFD_RELOC_SH_IMM4	76
BFD_RELOC_SCORE_IMM32	102	BFD_RELOC_SH_IMM4BY2	76
BFD_RELOC_SCORE_JMP	102	BFD_RELOC_SH_IMM4BY4	76
BFD_RELOC_SCORE16_BRANCH	102	BFD_RELOC_SH_IMM8	76
BFD_RELOC_SCORE16_JMP	102	BFD_RELOC_SH_IMM8BY2	76
BFD_RELOC_SH_ALIGN	76	BFD_RELOC_SH_IMM8BY4	76
BFD_RELOC_SH_CODE	76	BFD_RELOC_SH_IMMS10	77
BFD_RELOC_SH_COPY	77	BFD_RELOC_SH_IMMS10BY2	77
BFD_RELOC_SH_COPY64	77	BFD_RELOC_SH_IMMS10BY4	77
BFD_RELOC_SH_COUNT	76	BFD_RELOC_SH_IMMS10BY8	77
BFD_RELOC_SH_DATA	76	BFD_RELOC_SH_IMMS16	77
BFD_RELOC_SH_DISP12	76	BFD_RELOC_SH_IMMS6	77
BFD_RELOC_SH_DISP12BY2	76	BFD_RELOC_SH_IMMS6BY32	77
BFD_RELOC_SH_DISP12BY4	76	BFD_RELOC_SH_IMMU16	77
BFD_RELOC_SH_DISP12BY8	76	BFD_RELOC_SH_IMMU5	77
BFD_RELOC_SH_DISP20	76	BFD_RELOC_SH_IMMU6	77
BFD_RELOC_SH_DISP20BY8	76	BFD_RELOC_SH_JMP_SLOT	77
BFD_RELOC_SH_FUNCDESC	78	BFD_RELOC_SH_JMP_SLOT64	77
BFD_RELOC_SH_GLOB_DAT	77	BFD_RELOC_SH_LABEL	76
BFD_RELOC_SH_GLOB_DAT64	77	BFD_RELOC_SH_LOOP_END	77
BFD_RELOC_SH_GOT_HI16	77	BFD_RELOC_SH_LOOP_START	76
BFD_RELOC_SH_GOT_LOW16	77	BFD_RELOC_SH_PCDISP12BY2	76
BFD_RELOC_SH_GOT_MEDHI16	77	BFD_RELOC_SH_PCDISP8BY2	76
BFD_RELOC_SH_GOT_MEDLOW16	77	BFD_RELOC_SH_PCRELIMM8BY2	76
BFD_RELOC_SH_GOT10BY4	77	BFD_RELOC_SH_PCRELIMM8BY4	76
BFD_RELOC_SH_GOT10BY8	77	BFD_RELOC_SH_PLT_HI16	77
BFD_RELOC_SH_GOT20	78	BFD_RELOC_SH_PLT_LOW16	77
BFD_RELOC_SH_GOTFUNCDESC	78	BFD_RELOC_SH_PLT_MEDHI16	77
BFD_RELOC_SH_GOTFUNCDESC20	78	BFD_RELOC_SH_PLT_MEDLOW16	77
BFD_RELOC_SH_GOTOFF_HI16	77	BFD_RELOC_SH_PT_16	78
BFD_RELOC_SH_GOTOFF_LOW16	77	BFD_RELOC_SH_RELATIVE	77
BFD_RELOC_SH_GOTOFF_MEDHI16	77	BFD_RELOC_SH_RELATIVE64	77
BFD_RELOC_SH_GOTOFF_MEDLOW16	77	BFD_RELOC_SH_SHMEDIA_CODE	77
BFD_RELOC_SH_GOTOFF20	78	BFD_RELOC_SH_SWITCH16	76
BFD_RELOC_SH_GOTOFFFUNCDESC	78	BFD_RELOC_SH_SWITCH32	76
BFD_RELOC_SH_GOTOFFFUNCDESC20	78	BFD_RELOC_SH_TLS_DTPMOD32	78
BFD_RELOC_SH_GOTPC	77	BFD_RELOC_SH_TLS_DTPOFF32	78
BFD_RELOC_SH_GOTPC_HI16	77	BFD_RELOC_SH_TLS_GD_32	78
BFD_RELOC_SH_GOTPC_LOW16	77	BFD_RELOC_SH_TLS_IE_32	78
BFD_RELOC_SH_GOTPC_MEDHI16	77	BFD_RELOC_SH_TLS_LD_32	78
BFD_RELOC_SH_GOTPC_MEDLOW16	77	BFD_RELOC_SH_TLS_LDO_32	78
BFD_RELOC_SH_GOTPLT_HI16	77	BFD_RELOC_SH_TLS_LE_32	78
BFD_RELOC_SH_GOTPLT_LOW16	77	BFD_RELOC_SH_TLS_TPOFF32	78
BFD_RELOC_SH_GOTPLT_MEDHI16	77	BFD_RELOC_SH_USES	76
BFD_RELOC_SH_GOTPLT_MEDLOW16	77	BFD_RELOC_SIZE32	56
BFD_RELOC_SH_GOTPLT10BY4	77	BFD_RELOC_SIZE64	56
BFD_RELOC_SH_GOTPLT10BY8	77	BFD_RELOC_SPARC_10	58
BFD_RELOC_SH_GOTPLT32	77	BFD_RELOC_SPARC_11	58
BFD_RELOC_SH_IMM_HI16	78	BFD_RELOC_SPARC_5	58
BFD_RELOC_SH_IMM_HI16_PCREL	78	BFD_RELOC_SPARC_6	58
BFD_RELOC_SH_IMM_LOW16	77	BFD_RELOC_SPARC_64	58
BFD_RELOC_SH_IMM_LOW16_PCREL	78	BFD_RELOC_SPARC_7	58
BFD_RELOC_SH_IMM_MEDHI16	78	BFD_RELOC_SPARC_BASE13	58
BFD_RELOC_SH_IMM_MEDHI16_PCREL	78	BFD_RELOC_SPARC_BASE22	58
BFD_RELOC_SH_IMM_MEDLOW16	78	BFD_RELOC_SPARC_COPY	58
BFD_RELOC_SH_IMM_MEDLOW16_PCREL	78	BFD_RELOC_SPARC_DISP64	58
BFD_RELOC_SH_IMM3	76	BFD_RELOC_SPARC_GLOB_DAT	58

BFD_RELOC_SPARC_GOT10	57	BFD_RELOC_SPARC_UA32	58
BFD_RELOC_SPARC_GOT13	58	BFD_RELOC_SPARC_UA64	58
BFD_RELOC_SPARC_GOT22	58	BFD_RELOC_SPARC_WDISP10	59
BFD_RELOC_SPARC_GOTDATA_HIX22	58	BFD_RELOC_SPARC_WDISP16	58
BFD_RELOC_SPARC_GOTDATA_LOX10	58	BFD_RELOC_SPARC_WDISP19	58
BFD_RELOC_SPARC_GOTDATA_OP	58	BFD_RELOC_SPARC_WDISP22	57
BFD_RELOC_SPARC_GOTDATA_OP_HIX22	58	BFD_RELOC_SPARC_WPLT30	58
BFD_RELOC_SPARC_GOTDATA_OP_LOX10	58	BFD_RELOC_SPARC13	57
BFD_RELOC_SPARC_H34	59	BFD_RELOC_SPARC22	57
BFD_RELOC_SPARC_H44	58	BFD_RELOC_SPU_ADD_PIC	60
BFD_RELOC_SPARC_HH22	58	BFD_RELOC_SPU_HI16	60
BFD_RELOC_SPARC_HIX22	58	BFD_RELOC_SPU_IMM10	59
BFD_RELOC_SPARC_HM10	58	BFD_RELOC_SPU_IMM10W	59
BFD_RELOC_SPARC_IRELATIVE	58	BFD_RELOC_SPU_IMM16	59
BFD_RELOC_SPARC_JMP_IREL	58	BFD_RELOC_SPU_IMM16W	59
BFD_RELOC_SPARC_JMP_SLOT	58	BFD_RELOC_SPU_IMM18	59
BFD_RELOC_SPARC_L44	59	BFD_RELOC_SPU_IMM7	59
BFD_RELOC_SPARC_LM22	58	BFD_RELOC_SPU_IMM8	59
BFD_RELOC_SPARC_LOX10	58	BFD_RELOC_SPU_LO16	59
BFD_RELOC_SPARC_M44	58	BFD_RELOC_SPU_PCREL16	59
BFD_RELOC_SPARC_OL010	58	BFD_RELOC_SPU_PCREL9a	59
BFD_RELOC_SPARC_PC_HH22	58	BFD_RELOC_SPU_PCREL9b	59
BFD_RELOC_SPARC_PC_HM10	58	BFD_RELOC_SPU_PPU32	60
BFD_RELOC_SPARC_PC_LM22	58	BFD_RELOC_SPU_PPU64	60
BFD_RELOC_SPARC_PC10	58	BFD_RELOC_THUMB_PCREL_BFCSEL	73
BFD_RELOC_SPARC_PC22	58	BFD_RELOC_THUMB_PCREL_BLX	72
BFD_RELOC_SPARC_PLT32	58	BFD_RELOC_THUMB_PCREL_BRANCH12	73
BFD_RELOC_SPARC_PLT64	58	BFD_RELOC_THUMB_PCREL_BRANCH20	73
BFD_RELOC_SPARC_REGISTER	59	BFD_RELOC_THUMB_PCREL_BRANCH23	73
BFD_RELOC_SPARC_RELATIVE	58	BFD_RELOC_THUMB_PCREL_BRANCH25	73
BFD_RELOC_SPARC_REV32	59	BFD_RELOC_THUMB_PCREL_BRANCH5	72
BFD_RELOC_SPARC_SIZE32	59	BFD_RELOC_THUMB_PCREL_BRANCH7	73
BFD_RELOC_SPARC_SIZE64	59	BFD_RELOC_THUMB_PCREL_BRANCH9	73
BFD_RELOC_SPARC_TLS_DTPMOD32	59	BFD_RELOC_TIC30_LDP	91
BFD_RELOC_SPARC_TLS_DTPMOD64	59	BFD_RELOC_TIC54X_16_OF_23	91
BFD_RELOC_SPARC_TLS_DTPOFF32	59	BFD_RELOC_TIC54X_23	91
BFD_RELOC_SPARC_TLS_DTPOFF64	59	BFD_RELOC_TIC54X_MS7_OF_23	91
BFD_RELOC_SPARC_TLS_GD_ADD	59	BFD_RELOC_TIC54X_PARTLS7	91
BFD_RELOC_SPARC_TLS_GD_CALL	59	BFD_RELOC_TIC54X_PARTMS9	91
BFD_RELOC_SPARC_TLS_GD_HI22	59	BFD_RELOC_TILEGX_BROFF_X1	129
BFD_RELOC_SPARC_TLS_GD_LO10	59	BFD_RELOC_TILEGX_COPY	129
BFD_RELOC_SPARC_TLS_IE_ADD	59	BFD_RELOC_TILEGX_DEST_IMM8_X1	129
BFD_RELOC_SPARC_TLS_IE_HI22	59	BFD_RELOC_TILEGX_GLOB_DAT	129
BFD_RELOC_SPARC_TLS_IE_LD	59	BFD_RELOC_TILEGX_HWO	129
BFD_RELOC_SPARC_TLS_IE_LDX	59	BFD_RELOC_TILEGX_HWO_LAST	129
BFD_RELOC_SPARC_TLS_IE_LO10	59	BFD_RELOC_TILEGX_HW1	129
BFD_RELOC_SPARC_TLS_LDM_ADD	59	BFD_RELOC_TILEGX_HW1_LAST	129
BFD_RELOC_SPARC_TLS_LDM_CALL	59	BFD_RELOC_TILEGX_HW2	129
BFD_RELOC_SPARC_TLS_LDM_HI22	59	BFD_RELOC_TILEGX_HW2_LAST	129
BFD_RELOC_SPARC_TLS_LDM_LO10	59	BFD_RELOC_TILEGX_HW3	129
BFD_RELOC_SPARC_TLS_LDO_ADD	59	BFD_RELOC_TILEGX_IMM16_XO_HWO	129
BFD_RELOC_SPARC_TLS_LDO_HIX22	59	BFD_RELOC_TILEGX_IMM16_XO_HWO_GOT	130
BFD_RELOC_SPARC_TLS_LDO_LOX10	59	BFD_RELOC_TILEGX_IMM16_XO_HWO_LAST	130
BFD_RELOC_SPARC_TLS_LE_HIX22	59	BFD_RELOC_TILEGX_IMM16_XO_HWO_LAST_GOT	130
BFD_RELOC_SPARC_TLS_LE_LOX10	59	BFD_RELOC_TILEGX_IMM16_XO_	
BFD_RELOC_SPARC_TLS_TPOFF32	59	HWO_LAST_PCREL	130
BFD_RELOC_SPARC_TLS_TPOFF64	59	BFD_RELOC_TILEGX_IMM16_XO_HWO_	
BFD_RELOC_SPARC_UA16	58	LAST_PLT_PCREL	131

BFD_RELOC_TILEGX_IMM16_X0_	
HWO_LAST_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HWO_LAST_TLS_IE.....	131
BFD_RELOC_TILEGX_IMM16_X0_	
HWO_LAST_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X0_HWO_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HWO_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_IE.....	130
BFD_RELOC_TILEGX_IMM16_X0_HWO_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW1.....	129
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW1_LAST_GOT... 130	
BFD_RELOC_TILEGX_IMM16_X0_	
HW1_LAST_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW1_	
LAST_PLT_PCREL.....	131
BFD_RELOC_TILEGX_IMM16_X0_	
HW1_LAST_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HW1_LAST_TLS_IE.....	131
BFD_RELOC_TILEGX_IMM16_X0_	
HW1_LAST_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW1_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HW1_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW2.....	129
BFD_RELOC_TILEGX_IMM16_X0_HW2_LAST.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HW2_LAST_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW2_	
LAST_PLT_PCREL.....	131
BFD_RELOC_TILEGX_IMM16_X0_HW2_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HW2_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_HW3.....	129
BFD_RELOC_TILEGX_IMM16_X0_HW3_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X0_	
HW3_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO.....	129
BFD_RELOC_TILEGX_IMM16_X1_HWO_GOT.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_LAST_GOT... 130	
BFD_RELOC_TILEGX_IMM16_X1_	
HWO_LAST_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_	
LAST_PLT_PCREL.....	131
BFD_RELOC_TILEGX_IMM16_X1_	
HWO_LAST_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HWO_LAST_TLS_IE.....	131
BFD_RELOC_TILEGX_IMM16_X1_	
HWO_LAST_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HWO_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_IE.....	131
BFD_RELOC_TILEGX_IMM16_X1_HWO_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW1.....	129
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW1_LAST_GOT... 130	
BFD_RELOC_TILEGX_IMM16_X1_	
HW1_LAST_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW1_	
LAST_PLT_PCREL.....	131
BFD_RELOC_TILEGX_IMM16_X1_	
HW1_LAST_TLS_GD.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HW1_LAST_TLS_IE.....	131
BFD_RELOC_TILEGX_IMM16_X1_	
HW1_LAST_TLS_LE.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW1_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HW1_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW2.....	129
BFD_RELOC_TILEGX_IMM16_X1_HW2_LAST.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HW2_LAST_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW2_	
LAST_PLT_PCREL.....	131
BFD_RELOC_TILEGX_IMM16_X1_HW2_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HW2_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_HW3.....	129
BFD_RELOC_TILEGX_IMM16_X1_HW3_PCREL.....	130
BFD_RELOC_TILEGX_IMM16_X1_	
HW3_PLT_PCREL.....	130
BFD_RELOC_TILEGX_IMM8_XO.....	129
BFD_RELOC_TILEGX_IMM8_XO_TLS_ADD.....	131
BFD_RELOC_TILEGX_IMM8_XO_TLS_GD_ADD.....	131
BFD_RELOC_TILEGX_IMM8_X1.....	129
BFD_RELOC_TILEGX_IMM8_X1_TLS_ADD.....	131
BFD_RELOC_TILEGX_IMM8_X1_TLS_GD_ADD.....	131
BFD_RELOC_TILEGX_IMM8_YO.....	129
BFD_RELOC_TILEGX_IMM8_YO_TLS_ADD.....	131
BFD_RELOC_TILEGX_IMM8_YO_TLS_GD_ADD.....	131
BFD_RELOC_TILEGX_IMM8_Y1.....	129
BFD_RELOC_TILEGX_IMM8_Y1_TLS_ADD.....	131
BFD_RELOC_TILEGX_IMM8_Y1_TLS_GD_ADD.....	131
BFD_RELOC_TILEGX_JMP_SLOT.....	129
BFD_RELOC_TILEGX_JUMPOFF_X1.....	129
BFD_RELOC_TILEGX_JUMPOFF_X1_PLT.....	129
BFD_RELOC_TILEGX_MF_IMM14_X1.....	129
BFD_RELOC_TILEGX_MMEND_XO.....	129
BFD_RELOC_TILEGX_MMSTART_XO.....	129
BFD_RELOC_TILEGX_MT_IMM14_X1.....	129
BFD_RELOC_TILEGX_RELATIVE.....	129
BFD_RELOC_TILEGX_SHAMT_XO.....	129
BFD_RELOC_TILEGX_SHAMT_X1.....	129
BFD_RELOC_TILEGX_SHAMT_YO.....	129
BFD_RELOC_TILEGX_SHAMT_Y1.....	129
BFD_RELOC_TILEGX_TLS_DTPMOD32.....	131
BFD_RELOC_TILEGX_TLS_DTPMOD64.....	131

BFD_RELOC_TILEGX_TLS_DTPOFF32	131	BFD_RELOC_TILEPRO_IMM8_X0	127
BFD_RELOC_TILEGX_TLS_DTPOFF64	131	BFD_RELOC_TILEPRO_IMM8_X0_TLS_GD_ADD	128
BFD_RELOC_TILEGX_TLS_GD_CALL	131	BFD_RELOC_TILEPRO_IMM8_X1	127
BFD_RELOC_TILEGX_TLS_IE_LOAD	131	BFD_RELOC_TILEPRO_IMM8_X1_TLS_GD_ADD	128
BFD_RELOC_TILEGX_TLS_TPOFF32	131	BFD_RELOC_TILEPRO_IMM8_Y0	127
BFD_RELOC_TILEGX_TLS_TPOFF64	131	BFD_RELOC_TILEPRO_IMM8_Y0_TLS_GD_ADD	128
BFD_RELOC_TILEPRO_BROFF_X1	127	BFD_RELOC_TILEPRO_IMM8_Y1	127
BFD_RELOC_TILEPRO_COPY	127	BFD_RELOC_TILEPRO_IMM8_Y1_TLS_GD_ADD	128
BFD_RELOC_TILEPRO_DEST_IMM8_X1	127	BFD_RELOC_TILEPRO_IMM8_SLOT	127
BFD_RELOC_TILEPRO_GLOB_DAT	127	BFD_RELOC_TILEPRO_JOFFLONG_X1	127
BFD_RELOC_TILEPRO_IMM16_X0	127	BFD_RELOC_TILEPRO_JOFFLONG_X1_PLT	127
BFD_RELOC_TILEPRO_IMM16_X0_GOT	128	BFD_RELOC_TILEPRO_MF_IMM15_X1	127
BFD_RELOC_TILEPRO_IMM16_X0_GOT_HA	128	BFD_RELOC_TILEPRO_MMEND_X0	128
BFD_RELOC_TILEPRO_IMM16_X0_GOT_HI	128	BFD_RELOC_TILEPRO_MMEND_X1	128
BFD_RELOC_TILEPRO_IMM16_X0_GOT_LO	128	BFD_RELOC_TILEPRO_MMSTART_X0	128
BFD_RELOC_TILEPRO_IMM16_X0_HA	127	BFD_RELOC_TILEPRO_MMSTART_X1	128
BFD_RELOC_TILEPRO_IMM16_X0_HA_PCREL	128	BFD_RELOC_TILEPRO_MT_IMM15_X1	127
BFD_RELOC_TILEPRO_IMM16_X0_HI	127	BFD_RELOC_TILEPRO_RELATIVE	127
BFD_RELOC_TILEPRO_IMM16_X0_HI_PCREL	128	BFD_RELOC_TILEPRO_SHAMT_X0	128
BFD_RELOC_TILEPRO_IMM16_X0_LO	127	BFD_RELOC_TILEPRO_SHAMT_X1	128
BFD_RELOC_TILEPRO_IMM16_X0_LO_PCREL	128	BFD_RELOC_TILEPRO_SHAMT_Y0	128
BFD_RELOC_TILEPRO_IMM16_X0_PCREL	128	BFD_RELOC_TILEPRO_SHAMT_Y1	128
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD	128	BFD_RELOC_TILEPRO_TLS_DTPMOD32	128
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_HA	128	BFD_RELOC_TILEPRO_TLS_DTPOFF32	129
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_HI	128	BFD_RELOC_TILEPRO_TLS_GD_CALL	128
BFD_RELOC_TILEPRO_IMM16_X0_TLS_GD_LO	128	BFD_RELOC_TILEPRO_TLS_IE_LOAD	128
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE	128	BFD_RELOC_TILEPRO_TLS_TPOFF32	129
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_HA	128	BFD_RELOC_V850_16_GOT	90
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_HI	128	BFD_RELOC_V850_16_GOTOFF	90
BFD_RELOC_TILEPRO_IMM16_X0_TLS_IE_LO	128	BFD_RELOC_V850_16_PCREL	89
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE	129	BFD_RELOC_V850_16_S1	90
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_HA	129	BFD_RELOC_V850_16_SPLIT_OFFSET	90
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_HI	129	BFD_RELOC_V850_17_PCREL	89
BFD_RELOC_TILEPRO_IMM16_X0_TLS_LE_LO	129	BFD_RELOC_V850_22_PCREL	88
BFD_RELOC_TILEPRO_IMM16_X1	127	BFD_RELOC_V850_22_PLT_PCREL	90
BFD_RELOC_TILEPRO_IMM16_X1_GOT	128	BFD_RELOC_V850_23	90
BFD_RELOC_TILEPRO_IMM16_X1_GOT_HA	128	BFD_RELOC_V850_32_ABS	90
BFD_RELOC_TILEPRO_IMM16_X1_GOT_HI	128	BFD_RELOC_V850_32_GOT	90
BFD_RELOC_TILEPRO_IMM16_X1_GOT_LO	128	BFD_RELOC_V850_32_GOTOFF	90
BFD_RELOC_TILEPRO_IMM16_X1_HA	127	BFD_RELOC_V850_32_GOTPCREL	90
BFD_RELOC_TILEPRO_IMM16_X1_HA_PCREL	128	BFD_RELOC_V850_32_PCREL	90
BFD_RELOC_TILEPRO_IMM16_X1_HI	127	BFD_RELOC_V850_32_PLT_PCREL	90
BFD_RELOC_TILEPRO_IMM16_X1_HI_PCREL	128	BFD_RELOC_V850_9_PCREL	88
BFD_RELOC_TILEPRO_IMM16_X1_LO	127	BFD_RELOC_V850_ALIGN	89
BFD_RELOC_TILEPRO_IMM16_X1_LO_PCREL	128	BFD_RELOC_V850_CALLT_15_16_OFFSET	90
BFD_RELOC_TILEPRO_IMM16_X1_PCREL	128	BFD_RELOC_V850_CALLT_16_16_OFFSET	89
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD	128	BFD_RELOC_V850_CALLT_6_7_OFFSET	89
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_HA	128	BFD_RELOC_V850_CODE	91
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_HI	128	BFD_RELOC_V850_COPY	90
BFD_RELOC_TILEPRO_IMM16_X1_TLS_GD_LO	128	BFD_RELOC_V850_DATA	91
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE	128	BFD_RELOC_V850_GLOB_DAT	90
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_HA	128	BFD_RELOC_V850_JMP_SLOT	90
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_HI	128	BFD_RELOC_V850_L016_S1	90
BFD_RELOC_TILEPRO_IMM16_X1_TLS_IE_LO	128	BFD_RELOC_V850_L016_SPLIT_OFFSET	89
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE	129	BFD_RELOC_V850_LONGCALL	89
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_HA	129	BFD_RELOC_V850_LONGJUMP	89
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_HI	129	BFD_RELOC_V850_RELATIVE	90
BFD_RELOC_TILEPRO_IMM16_X1_TLS_LE_LO	129	BFD_RELOC_V850_SDA_15_16_OFFSET	88

BFD_RELOC_V850_SDA_16_16_OFFSET	88	BFD_RELOC_X86_64_RELATIVE	67
BFD_RELOC_V850_SDA_16_16_SPLIT_OFFSET	89	BFD_RELOC_X86_64_REX_GOTPCRELX	68
BFD_RELOC_V850_TDA_16_16_OFFSET	89	BFD_RELOC_X86_64_TLSDESC	68
BFD_RELOC_V850_TDA_4_4_OFFSET	89	BFD_RELOC_X86_64_TLSDESC_CALL	68
BFD_RELOC_V850_TDA_4_5_OFFSET	89	BFD_RELOC_X86_64_TLSD	67
BFD_RELOC_V850_TDA_6_8_OFFSET	89	BFD_RELOC_X86_64_TLSD	67
BFD_RELOC_V850_TDA_7_7_OFFSET	89	BFD_RELOC_X86_64_TPOFF32	67
BFD_RELOC_V850_TDA_7_8_OFFSET	89	BFD_RELOC_X86_64_TPOFF64	67
BFD_RELOC_V850_ZDA_15_16_OFFSET	89	BFD_RELOC_XGATE_24	106
BFD_RELOC_V850_ZDA_16_16_OFFSET	88	BFD_RELOC_XGATE_GPAGE	106
BFD_RELOC_V850_ZDA_16_16_SPLIT_OFFSET	89	BFD_RELOC_XGATE_IMM3	106
BFD_RELOC_VAX_GLOB_DAT	110	BFD_RELOC_XGATE_IMM4	106
BFD_RELOC_VAX_JMP_SLOT	110	BFD_RELOC_XGATE_IMM5	106
BFD_RELOC_VAX_RELATIVE	110	BFD_RELOC_XGATE_IMM8_HI	106
BFD_RELOC_VISIUM_HI16	132	BFD_RELOC_XGATE_IMM8_LO	106
BFD_RELOC_VISIUM_HI16_PCREL	132	BFD_RELOC_XGATE_LO16	106
BFD_RELOC_VISIUM_IM16	132	BFD_RELOC_XGATE_PCREL_10	106
BFD_RELOC_VISIUM_IM16_PCREL	132	BFD_RELOC_XGATE_PCREL_9	106
BFD_RELOC_VISIUM_LO16	132	BFD_RELOC_XGATE_RL_GROUP	106
BFD_RELOC_VISIUM_LO16_PCREL	132	BFD_RELOC_XGATE_RL_JUMP	106
BFD_RELOC_VISIUM_REL16	132	BFD_RELOC_XSTORMY16_12	110
BFD_RELOC_VPE4KMATH_DATA	103	BFD_RELOC_XSTORMY16_24	110
BFD_RELOC_VPE4KMATH_INSN	103	BFD_RELOC_XSTORMY16_FPTR16	110
BFD_RELOC_VTABLE_ENTRY	103	BFD_RELOC_XSTORMY16_REL_12	110
BFD_RELOC_VTABLE_INHERIT	103	BFD_RELOC_XTENZA_ASM_EXPAND	115
BFD_RELOC_WASM32_ABS32_CODE	132	BFD_RELOC_XTENZA_ASM_SIMPLIFY	115
BFD_RELOC_WASM32_CODE_POINTER	132	BFD_RELOC_XTENZA_DIFF16	114
BFD_RELOC_WASM32_COPY	132	BFD_RELOC_XTENZA_DIFF32	114
BFD_RELOC_WASM32_INDEX	132	BFD_RELOC_XTENZA_DIFF8	114
BFD_RELOC_WASM32_LEB128	132	BFD_RELOC_XTENZA_GLOB_DAT	113
BFD_RELOC_WASM32_LEB128_GOT	132	BFD_RELOC_XTENZA_JMP_SLOT	113
BFD_RELOC_WASM32_LEB128_GOT_CODE	132	BFD_RELOC_XTENZA_NDIFF16	115
BFD_RELOC_WASM32_LEB128_PLT	132	BFD_RELOC_XTENZA_NDIFF32	115
BFD_RELOC_WASM32_PLT_INDEX	132	BFD_RELOC_XTENZA_NDIFF8	115
BFD_RELOC_WASM32_PLT_SIG	132	BFD_RELOC_XTENZA_OP0	115
BFD_RELOC_X86_64_32S	67	BFD_RELOC_XTENZA_OP1	115
BFD_RELOC_X86_64_COPY	67	BFD_RELOC_XTENZA_OP2	115
BFD_RELOC_X86_64_DTPMOD64	67	BFD_RELOC_XTENZA_PDIFF16	115
BFD_RELOC_X86_64_DTPOFF32	67	BFD_RELOC_XTENZA_PDIFF32	115
BFD_RELOC_X86_64_DTPOFF64	67	BFD_RELOC_XTENZA_PDIFF8	115
BFD_RELOC_X86_64_GLOB_DAT	67	BFD_RELOC_XTENZA_PLT	114
BFD_RELOC_X86_64_GOT32	67	BFD_RELOC_XTENZA_RELATIVE	113
BFD_RELOC_X86_64_GOT64	67	BFD_RELOC_XTENZA_RTLD	113
BFD_RELOC_X86_64_GOTOFF64	67	BFD_RELOC_XTENZA_SLOT0_ALT	114
BFD_RELOC_X86_64_GOTPC32	67	BFD_RELOC_XTENZA_SLOT0_OP	114
BFD_RELOC_X86_64_GOTPC32_TLSDESC	68	BFD_RELOC_XTENZA_SLOT1_ALT	114
BFD_RELOC_X86_64_GOTPC64	67	BFD_RELOC_XTENZA_SLOT1_OP	114
BFD_RELOC_X86_64_GOTPCREL	67	BFD_RELOC_XTENZA_SLOT10_ALT	114
BFD_RELOC_X86_64_GOTPCREL64	67	BFD_RELOC_XTENZA_SLOT10_OP	114
BFD_RELOC_X86_64_GOTPCRELX	68	BFD_RELOC_XTENZA_SLOT11_ALT	114
BFD_RELOC_X86_64_GOTPLT64	68	BFD_RELOC_XTENZA_SLOT11_OP	114
BFD_RELOC_X86_64_GOTPOFF	67	BFD_RELOC_XTENZA_SLOT12_ALT	114
BFD_RELOC_X86_64_IRELATIVE	68	BFD_RELOC_XTENZA_SLOT12_OP	114
BFD_RELOC_X86_64_JUMP_SLOT	67	BFD_RELOC_XTENZA_SLOT13_ALT	114
BFD_RELOC_X86_64_PC32_BND	68	BFD_RELOC_XTENZA_SLOT13_OP	114
BFD_RELOC_X86_64_PLT32	67	BFD_RELOC_XTENZA_SLOT14_ALT	114
BFD_RELOC_X86_64_PLT32_BND	68	BFD_RELOC_XTENZA_SLOT14_OP	114
BFD_RELOC_X86_64_PLTOFF64	68	BFD_RELOC_XTENZA_SLOT2_ALT	114

BFD_RELOC_XTENSA_SLOT2_OP 114
 BFD_RELOC_XTENSA_SLOT3_ALT 114
 BFD_RELOC_XTENSA_SLOT3_OP 114
 BFD_RELOC_XTENSA_SLOT4_ALT 114
 BFD_RELOC_XTENSA_SLOT4_OP 114
 BFD_RELOC_XTENSA_SLOT5_ALT 114
 BFD_RELOC_XTENSA_SLOT5_OP 114
 BFD_RELOC_XTENSA_SLOT6_ALT 114
 BFD_RELOC_XTENSA_SLOT6_OP 114
 BFD_RELOC_XTENSA_SLOT7_ALT 114
 BFD_RELOC_XTENSA_SLOT7_OP 114
 BFD_RELOC_XTENSA_SLOT8_ALT 114
 BFD_RELOC_XTENSA_SLOT8_OP 114
 BFD_RELOC_XTENSA_SLOT9_ALT 114
 BFD_RELOC_XTENSA_SLOT9_OP 114
 BFD_RELOC_XTENSA_TLS_ARG 115
 BFD_RELOC_XTENSA_TLS_CALL 115
 BFD_RELOC_XTENSA_TLS_DTPOFF 115
 BFD_RELOC_XTENSA_TLS_FUNC 115
 BFD_RELOC_XTENSA_TLS_TPOFF 115
 BFD_RELOC_XTENSA_TLSDESC_ARG 115
 BFD_RELOC_XTENSA_TLSDESC_FN 115
 BFD_RELOC_Z80_16_BE 116
 BFD_RELOC_Z80_BYTE0 115
 BFD_RELOC_Z80_BYTE1 115
 BFD_RELOC_Z80_BYTE2 116
 BFD_RELOC_Z80_BYTE3 116
 BFD_RELOC_Z80_DISP8 115
 BFD_RELOC_Z80_WORD0 116
 BFD_RELOC_Z80_WORD1 116
 BFD_RELOC_Z8K_CALLR 116
 BFD_RELOC_Z8K_DISP7 116
 BFD_RELOC_Z8K_IMM4L 116

C

char ... 13, 14, 38, 48, 136, 137, 151, 163, 165, 172
 coff_symbol_type 190
 core_file_matches_executable_p 137

G

generic_core_file_matches_executable_p 138

H

Hash tables 179

I

int 14, 16, 164, 165, 183
 internal object-file format 3

L

Linker 173
 long 164

M

machine_type 187

O

Other functions 17

P

per_xvec_message 150

S

struct bfd_iovec 20

T

target vector (_bfd_final_link) 175
 target vector (_bfd_link_add_symbols) 173
 target vector (_bfd_link_hash_table_create) 173
 The HOWTO Macro 53

W

what is it? 1

The body of this manual is set in
cmr10,
with headings in **cmbx10**
and examples in cmtt10.
cmti10 and
cmsl10
are used for emphasis.