

This is as.info, produced by makeinfo version 6.3 from as.texinfo.

This file documents the GNU Assembler "as".

Copyright (C) 1991-2017 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".

INFO-DIR-SECTION Software development

START-INFO-DIR-ENTRY

* As: (as). The GNU assembler.

* Gas: (as). The GNU assembler.

END-INFO-DIR-ENTRY



File: as.info, Node: Top, Next: Overview, Up: (dir)

Using as

This file is a user guide to the GNU assembler 'as' (GNU Binutils) version 2.28.

This document is distributed under the terms of the GNU Free Documentation License. A copy of the license is included in the section entitled "GNU Free Documentation License".

* Menu:

* Overview::	Overview
* Invoking::	Command-Line Options
* Syntax::	Syntax
* Sections::	Sections and Relocation
* Symbols::	Symbols
* Expressions::	Expressions
* Pseudo Ops::	Assembler Directives
* Object Attributes::	Object Attributes
* Machine Dependencies::	Machine Dependent Features
* Reporting Bugs::	Reporting Bugs
* Acknowledgements::	Who Did What
* GNU Free Documentation License::	GNU Free Documentation License
* AS Index::	AS Index



File: as.info, Node: Overview, Next: Invoking, Prev: Top, Up: Top

1 Overview

Here is a brief summary of how to invoke 'as'. For details, see *note Command-Line Options: Invoking.

```
as [-a[cdghlns][=FILE]] [-alternate] [-D]
   [-compress-debug-sections] [-nocompress-debug-sections]
```

```

[-debug-prefix-map OLD=NEW]
[-defsym SYM=VAL] [-f] [-g] [-gstabs]
[-gstabs+] [-gdwarf-2] [-gdwarf-sections]
[-help] [-I DIR] [-J]
[-K] [-L] [-listing-lhs-width=NUM]
[-listing-lhs-width2=NUM] [-listing-rhs-width=NUM]
[-listing-cont-lines=NUM] [-keep-locals]
[-no-pad-sections]
[-o OBJFILE] [-R]
[-hash-size=NUM] [-reduce-memory-overheads]
[-statistics]
[-v] [-version] [-version]
[-W] [-warn] [-fatal-warnings] [-w] [-x]
[-Z] [@FILE]
[-sectname-subst] [-size-check=[error|warning]]
[-elf-stt-common=[no|yes]]
[-target-help] [TARGET-OPTIONS]
[-|FILES ...]

```

Target AArch64 options:

```

[-EB|-EL]
[-mabi=ABI]

```

Target Alpha options:

```

[-mCPU]
[-mdebug | -no-mdebug]
[-replace | -noreplace]
[-relax] [-g] [-GSIZE]
[-F] [-32addr]

```

Target ARC options:

```

[-mcpu=CPU]
[-mA6|-mARC600|-mARC601|-mA7|-mARC700|-mEM|-mHS]
[-mcode-density]
[-mrelax]
[-EB|-EL]

```

Target ARM options:

```

[-mcpu=PROCESSOR[+EXTENSION...]]
[-march=ARCHITECTURE[+EXTENSION...]]
[-mfpu=FLOATING-POINT-FORMAT]
[-mfloat-abi=ABI]
[-meabi=VER]
[-mthumb]
[-EB|-EL]
[-mapcs-32|-mapcs-26|-mapcs-float|
-mapcs-reentrant]
[-mthumb-interwork] [-k]

```

Target Blackfin options:

```

[-mcpu=PROCESSOR[-SIREVISION]]
[-mfdpic]
[-mno-fdpic]
[-mnopic]

```

Target CRIS options:

```

[-underscore | -no-underscore]
[-pic] [-N]
[-emulation=criself | -emulation=crisaout]

```

```
[-march=v0_v10 | -march=v10 | -march=v32 | -march=common_v10_v32]
```

```
_Target D10V options: _
```

```
[-O]
```

```
_Target D30V options: _
```

```
[-O|-n|-N]
```

```
_Target EIPHANY options: _
```

```
[-mepiphany|-mepiphany16]
```

```
_Target H8/300 options: _
```

```
[-h-tick-hex]
```

```
_Target i386 options: _
```

```
[-32|-x32|-64] [-n]
```

```
[-march=CPU[+EXTENSION...]] [-mtune=CPU]
```

```
_Target i960 options: _
```

```
[-ACA|-ACA_A|-ACB|-ACC|-AKA|-AKB|
```

```
-AKC|-AMC]
```

```
[-b] [-no-relax]
```

```
_Target IA-64 options: _
```

```
[-mconstant-gp|-mauto-pic]
```

```
[-milp32|-milp64|-mlp64|-mp64]
```

```
[-mle|mbe]
```

```
[-mtune=itanium1|-mtune=itanium2]
```

```
[-munwind-check=warning|-munwind-check=error]
```

```
[-mhint.b=ok|-mhint.b=warning|-mhint.b=error]
```

```
[-x|-xexplicit] [-xauto] [-xdebug]
```

```
_Target IP2K options: _
```

```
[-mip2022|-mip2022ext]
```

```
_Target M32C options: _
```

```
[-m32c|-m16c] [-relax] [-h-tick-hex]
```

```
_Target M32R options: _
```

```
[-m32rx|-[no-]warn-explicit-parallel-conflicts|
```

```
-W[n]p]
```

```
_Target M680X0 options: _
```

```
[-l] [-m68000|-m68010|-m68020|...]
```

```
_Target M68HC11 options: _
```

```
[-m68hc11|-m68hc12|-m68hcs12|-mm9s12x|-mm9s12xg]
```

```
[-mshort|-mlong]
```

```
[-mshort-double|-mlong-double]
```

```
[-force-long-branches] [-short-branches]
```

```
[-strict-direct-mode] [-print-insn-syntax]
```

```
[-print-opcodes] [-generate-example]
```

```
_Target MCORE options: _
```

```
[-jsri2bsr] [-sifilter] [-relax]
```

```
[-mcpu=[210|340]]
```

```
_Target Meta options: _
```

```
[-mcpu=CPU] [-mfpu=CPU] [-mdsp=CPU]
```

Target MICROBLAZE options:

Target MIPS options:

```
[-nocpp] [-EL] [-EB] [-O[OPTIMIZATION LEVEL]]
[-g[DEBUG LEVEL]] [-G NUM] [-KPIC] [-call_shared]
[-non_shared] [-xgot [-mvxworks-pic]
[-mabi=ABI] [-32] [-n32] [-64] [-mfp32] [-mfp32]
[-mfp64] [-mfp64] [-mfp64] [-mfp64]
[-modd-spreg] [-mno-odd-spreg]
[-march=CPU] [-mtune=CPU] [-mips1] [-mips2]
[-mips3] [-mips4] [-mips5] [-mips32] [-mips32r2]
[-mips32r3] [-mips32r5] [-mips32r6] [-mips64] [-mips64r2]
[-mips64r3] [-mips64r5] [-mips64r6]
[-construct-floats] [-no-construct-floats]
[-mignore-branch-isa] [-mno-ignore-branch-isa]
[-mnan=ENCODING]
[-trap] [-no-break] [-break] [-no-trap]
[-mips16] [-no-mips16]
[-mmicromips] [-mno-micromips]
[-msmartmips] [-mno-smartmips]
[-mips3d] [-no-mips3d]
[-mdmx] [-no-mdmx]
[-mdsp] [-mno-dsp]
[-mdspr2] [-mno-dspr2]
[-mdspr3] [-mno-dspr3]
[-mmsa] [-mno-msa]
[-mxpa] [-mno-xpa]
[-mmt] [-mno-mt]
[-mmcui] [-mno-mcui]
[-minsn32] [-mno-insn32]
[-mfix7000] [-mno-fix7000]
[-mfix-rm7000] [-mno-fix-rm7000]
[-mfix-vr4120] [-mno-fix-vr4120]
[-mfix-vr4130] [-mno-fix-vr4130]
[-mdebug] [-no-mdebug]
[-mpdr] [-mno-pdr]
```

Target MMIX options:

```
[-fixed-special-register-names] [-globalize-symbols]
[-gnu-syntax] [-relax] [-no-predefined-symbols]
[-no-expand] [-no-merge-gregs] [-x]
[-linker-allocated-gregs]
```

Target Nios II options:

```
[-relax-all] [-relax-section] [-no-relax]
[-EB] [-EL]
```

Target NDS32 options:

```
[-EL] [-EB] [-O] [-Os] [-mcpu=CPU]
[-misa=ISA] [-mabi=ABI] [-mall-ext]
[-m[no-]16-bit] [-m[no-]perf-ext] [-m[no-]perf2-ext]
[-m[no-]string-ext] [-m[no-]dsp-ext] [-m[no-]mac] [-m[no-]div]
[-m[no-]audio-isa-ext] [-m[no-]fpu-sp-ext] [-m[no-]fpu-dp-ext]
[-m[no-]fpu-fma] [-mfpu-freg=FREG] [-mreduced-regs]
[-mfull-regs] [-m[no-]dx-regs] [-mpic] [-mno-relax]
[-mb2bb]
```

Target PDP11 options:

```
[-mpic|-mno-pic] [-mall] [-mno-extensions]
```

[-mEXTENSION| -mno-EXTENSION]
 [-mCPU] [-mMACHINE]

Target picoJava options:
 [-mb| -me]

Target PowerPC options:
 [-a32| -a64]
 [-mpwrx| -mpwr2| -mpwr| -m601| -mppc| -mppc32| -m603| -m604| -m403| -m405|
 -m440| -m464| -m476| -m7400| -m7410| -m7450| -m7455| -m750c1| -mppc64|
 -m620| -me500| -e500x2| -me500mc| -me500mc64| -me5500| -me6500| -mppc64bridge|
 -mbooke| -mpower4| -mpwr4| -mpower5| -mpwr5| -mpwr5x| -mpower6| -mpwr6|
 -mpower7| -mpwr7| -mpower8| -mpwr8| -mpower9| -mpwr9- ma2|
 -mcell| -mspe| -mtitan| -me300| -mcom]
 [-many] [-maltivec| -mvsx| -mhtm| -mvle]
 [-mregnames| -mno-regnames]
 [-mrelocatable| -mrelocatable-lib| -K PIC] [-memb]
 [-mlittle| -mlittle-endian| -le| -mbig| -mbig-endian| -be]
 [-msolaris| -mno-solaris]
 [-nops=COUNT]

Target RL78 options:
 [-mg10]
 [-m32bit-doubles| -m64bit-doubles]

Target RX options:
 [-mlittle-endian| -mbig-endian]
 [-m32bit-doubles| -m64bit-doubles]
 [-muse-conventional-section-names]
 [-msmall-data-limit]
 [-mpid]
 [-mrelax]
 [-mint-register=NUMBER]
 [-mgcc-abi| -mrx-abi]

Target RISC-V options:
 [-march=ISA]
 [-mabi=ABI]

Target s390 options:
 [-m31| -m64] [-mesa| -mzarch] [-march=CPU]
 [-mregnames| -mno-regnames]
 [-mwarn-areg-zero]

Target SCORE options:
 [-EB] [-EL] [-FIXDD] [-NWARN]
 [-SCORE5] [-SCORE5U] [-SCORE7] [-SCORE3]
 [-march=score7] [-march=score3]
 [-USE_R1] [-KPIC] [-00] [-G NUM] [-V]

Target SPARC options:
 [-Av6| -Av7| -Av8| -Aleon| -Asparclet| -Asparclite
 -Av8plus| -Av8plusa| -Av8plusb| -Av8plusc| -Av8plused
 -Av8plusv| -Av8plusm| -Av9| -Av9a| -Av9b| -Av9c
 -Av9d| -Av9e| -Av9v| -Av9m| -Asparc| -Asparcvis
 -Asparcvis2| -Asparcfmaf| -Asparcima| -Asparcvis3
 -Asparcvisr| -Asparc5]
 [-xarch=v8plus| -xarch=v8plusa] | -xarch=v8plusb| -xarch=v8plusc
 -xarch=v8plused| -xarch=v8plusv| -xarch=v8plusm| -xarch=v9

```
-xarch=v9a|-xarch=v9b|-xarch=v9c|-xarch=v9d|-xarch=v9e
-xarch=v9v|-xarch=v9m|-xarch=sparc|-xarch=sparcvis
-xarch=sparcvis2|-xarch=sparcfmaf|-xarch=sparcima
-xarch=sparcvis3|-xarch=sparcvisr|-xarch=sparc5
-bump]
[-32|-64]
[-enforce-aligned-data][-dcti-couples-detect]
```

Target TIC54X options:

```
[-mcpu=54[123589]|-mcpu=54[56]lp] [-mfar-mode|-mf]
[-merrors-to-file <FILENAME>|-me <FILENAME>]
```

Target TIC6X options:

```
[-march=ARCH] [-mbig-endian|-mlittle-endian]
[-mdsbt|-mno-dsbt] [-mpid=no|-mpid=near|-mpid=far]
[-mpic|-mno-pic]
```

Target TILE-Gx options:

```
[-m32|-m64][-EB][-EL]
```

Target Visium options:

```
[-mtune=ARCH]
```

Target Xtensa options:

```
[-[no-]text-section-literals] [-[no-]auto-litpools]
[-[no-]absolute-literals]
[-[no-]target-align] [-[no-]longcalls]
[-[no-]transform]
[-rename-section OLDNAME=NEWNAME]
[-[no-]trampolines]
```

Target Z80 options:

```
[-z80] [-r800]
[ -ignore-undocumented-instructions] [-Wnud]
[ -ignore-unportable-instructions] [-Wnup]
[ -warn-undocumented-instructions] [-Wud]
[ -warn-unportable-instructions] [-Wup]
[ -forbid-undocumented-instructions] [-Fud]
[ -forbid-unportable-instructions] [-Fup]
```

'@FILE'

Read command-line options from FILE. The options read are inserted in place of the original @FILE option. If FILE does not exist, or cannot be read, then the option will be treated literally, and not removed.

Options in FILE are separated by whitespace. A whitespace character may be included in an option by surrounding the entire option in either single or double quotes. Any character (including a backslash) may be included by prefixing the character to be included with a backslash. The FILE may itself contain additional @FILE options; any such options will be processed recursively.

'-a[cdghlmns]'

Turn on listings, in any of a variety of ways:

'-ac'

omit false conditionals

'-ad'

omit debugging directives

'-ag'

include general information, like as version and options passed

'-ah'

include high-level source

'-al'

include assembly

'-am'

include macro expansions

'-an'

omit forms processing

'-as'

include symbols

'=file'

set the name of the listing file

You may combine these options; for example, use '-aln' for assembly listing without forms processing. The '=file' option, if used, must be the last one. By itself, '-a' defaults to '-ahls'.

'--alternate'

Begin in alternate macro mode. *Note '.altmacro': Altmacro.

'--compress-debug-sections'

Compress DWARF debug sections using zlib with SHF_COMPRESSED from the ELF ABI. The resulting object file may not be compatible with older linkers and object file utilities. Note if compression would make a given section `_larger_` then it is not compressed.

'--compress-debug-sections=none'

'--compress-debug-sections=zlib'

'--compress-debug-sections=zlib-gnu'

'--compress-debug-sections=zlib-gabi'

These options control how DWARF debug sections are compressed.

'--compress-debug-sections=none' is equivalent to

'--nocompress-debug-sections'. '--compress-debug-sections=zlib'

and '--compress-debug-sections=zlib-gabi' are equivalent to

'--compress-debug-sections'. '--compress-debug-sections=zlib-gnu'

compresses DWARF debug sections using zlib. The debug sections are renamed to begin with '.zdebug'. Note if compression would make a given section `_larger_` then it is not compressed nor renamed.

'--nocompress-debug-sections'

Do not compress DWARF debug sections. This is usually the default for all targets except the x86/x86_64, but a configure time option can be used to override this.

'-D'

Ignored. This option is accepted for script compatibility with calls to other assemblers.

'--debug-prefix-map OLD=NEW'

When assembling files in directory 'OLD', record debugging information describing them as in 'NEW' instead.

'--defsym SYM=VALUE'

Define the symbol SYM to be VALUE before assembling the input file. VALUE must be an integer constant. As in C, a leading '0x' indicates a hexadecimal value, and a leading '0' indicates an octal value. The value of the symbol can be overridden inside a source file via the use of a '.set' pseudo-op.

'-f'

"fast"--skip whitespace and comment preprocessing (assume source is compiler output).

'-g'

'--gen-debug'

Generate debugging information for each assembler source line using whichever debug format is preferred by the target. This currently means either STABS, ECOFF or DWARF2.

'--gstabs'

Generate stabs debugging information for each assembler line. This may help debugging assembler code, if the debugger can handle it.

'--gstabs+'

Generate stabs debugging information for each assembler line, with GNU extensions that probably only gdb can handle, and that could make other debuggers crash or refuse to read your program. This may help debugging assembler code. Currently the only GNU extension is the location of the current working directory at assembling time.

'--gdwarf-2'

Generate DWARF2 debugging information for each assembler line. This may help debugging assembler code, if the debugger can handle it. Note--this option is only supported by some targets, not all of them.

'--gdwarf-sections'

Instead of creating a .debug_line section, create a series of .debug_line.FOO sections where FOO is the name of the corresponding code section. For example a code section called .TEXT.FUNC will have its dwarf line number information placed into a section called .DEBUG_LINE.TEXT.FUNC. If the code section is just called .TEXT then debug line section will still be called just .DEBUG_LINE without any suffix.

'--size-check=error'

'--size-check=warning'

Issue an error or warning for invalid ELF .size directive.

'--elf-stt-common=no'

'--elf-stt-common=yes'

These options control whether the ELF assembler should generate common symbols with the 'STT_COMMON' type. The default can be

controlled by a configure option '--enable-elf-stt-common'.

'--help'

Print a summary of the command line options and exit.

'--target-help'

Print a summary of all target specific options and exit.

'-I DIR'

Add directory DIR to the search list for '.include' directives.

'-J'

Don't warn about signed overflow.

'-K'

Issue warnings when difference tables altered for long displacements.

'-L'

'--keep-locals'

Keep (in the symbol table) local symbols. These symbols start with system-specific local label prefixes, typically '.L' for ELF systems or 'L' for traditional a.out systems. *Note Symbol Names:..

'--listing-lhs-width=NUMBER'

Set the maximum width, in words, of the output data column for an assembler listing to NUMBER.

'--listing-lhs-width2=NUMBER'

Set the maximum width, in words, of the output data column for continuation lines in an assembler listing to NUMBER.

'--listing-rhs-width=NUMBER'

Set the maximum width of an input source line, as displayed in a listing, to NUMBER bytes.

'--listing-cont-lines=NUMBER'

Set the maximum number of lines printed in a listing for a single line of input to NUMBER + 1.

'--no-pad-sections'

Stop the assembler for padding the ends of output sections to the alignment of that section. The default is to pad the sections, but this can waste space which might be needed on targets which have tight memory constraints.

'-o OBJFILE'

Name the object-file output from 'as' OBJFILE.

'-R'

Fold the data section into the text section.

'--hash-size=NUMBER'

Set the default size of GAS's hash tables to a prime number close to NUMBER. Increasing this value can reduce the length of time it takes the assembler to perform its tasks, at the expense of increasing the assembler's memory requirements. Similarly reducing this value can reduce the memory requirements at the expense of

speed.

'--reduce-memory-overheads'

This option reduces GAS's memory requirements, at the expense of making the assembly processes slower. Currently this switch is a synonym for '--hash-size=4051', but in the future it may have other effects as well.

'--sectname-subst'

Honor substitution sequences in section names. *Note '.section NAME': Section Name Substitutions.

'--statistics'

Print the maximum space (in bytes) and total time (in seconds) used by assembly.

'--strip-local-absolute'

Remove local absolute symbols from the outgoing symbol table.

'-v'

'-version'

Print the 'as' version.

'--version'

Print the 'as' version and exit.

'-W'

'--no-warn'

Suppress warning messages.

'--fatal-warnings'

Treat warnings as errors.

'--warn'

Don't suppress warning messages or treat them as errors.

'-w'

Ignored.

'-x'

Ignored.

'-Z'

Generate an object file even after errors.

'-- | FILES ...'

Standard input, or source files to assemble.

*Note AArch64 Options::, for the options available when as is configured for the 64-bit mode of the ARM Architecture (AArch64).

*Note Alpha Options::, for the options available when as is configured for an Alpha processor.

The following options are available when as is configured for an ARC processor.

'-mcpu=CPU'

This option selects the core processor variant.

- '-EB | -EL'
Select either big-endian (-EB) or little-endian (-EL) output.
- '-mcode-density'
Enable Code Density extension instructions.

The following options are available when as is configured for the ARM processor family.

- '-mcpu=PROCESSOR[+EXTENSION...]'
Specify which ARM processor variant is the target.
- '-march=ARCHITECTURE[+EXTENSION...]'
Specify which ARM architecture variant is used by the target.
- '-mfpu=FLOATING-POINT-FORMAT'
Select which Floating Point architecture is the target.
- '-mfloat-abi=ABI'
Select which floating point ABI is in use.
- '-mthumb'
Enable Thumb only instruction decoding.
- '-mapcs-32 | -mapcs-26 | -mapcs-float | -mapcs-reentrant'
Select which procedure calling convention is in use.
- '-EB | -EL'
Select either big-endian (-EB) or little-endian (-EL) output.
- '-mthumb-interwork'
Specify that the code has been generated with interworking between Thumb and ARM code in mind.
- '-mccs'
Turns on CodeComposer Studio assembly syntax compatibility mode.
- '-k'
Specify that PIC code has been generated.

*Note Blackfin Options::, for the options available when as is configured for the Blackfin processor family.

See the info pages for documentation of the CRIS-specific options.

The following options are available when as is configured for a D10V processor.

- '-O'
Optimize output by parallelizing instructions.

The following options are available when as is configured for a D30V processor.

- '-O'
Optimize output by parallelizing instructions.
- '-n'
Warn when nops are generated.
- '-N'
Warn when a nop after a 32-bit multiply instruction is generated.

The following options are available when as is configured for the Adapteva EPIPHANY series.

*Note Epiphany Options::, for the options available when as is configured for an Epiphany processor.

*Note i386-Options::, for the options available when as is configured for an i386 processor.

The following options are available when as is configured for the Intel 80960 processor.

'-ACA | -ACA_A | -ACB | -ACC | -AKA | -AKB | -AKC | -AMC'
Specify which variant of the 960 architecture is the target.

'-b'
Add code to collect statistics about branches taken.

'-no-relax'
Do not alter compare-and-branch instructions for long displacements; error if necessary.

The following options are available when as is configured for the Ubicom IP2K series.

'-mip2022ext'
Specifies that the extended IP2022 instructions are allowed.

'-mip2022'
Restores the default behaviour, which restricts the permitted instructions to just the basic IP2022 ones.

The following options are available when as is configured for the Renesas M32C and M16C processors.

'-m32c'
Assemble M32C instructions.

'-m16c'
Assemble M16C instructions (the default).

'-relax'
Enable support for link-time relaxations.

'-h-tick-hex'
Support H'00 style hex constants in addition to 0x00 style.

The following options are available when as is configured for the Renesas M32R (formerly Mitsubishi M32R) series.

'--m32rx'
Specify which processor in the M32R family is the target. The default is normally the M32R, but this option changes it to the M32RX.

'--warn-explicit-parallel-conflicts or --Wp'
Produce warning messages when questionable parallel constructs are encountered.

'--no-warn-explicit-parallel-conflicts or --Wnp'
Do not produce warning messages when questionable parallel constructs are encountered.

The following options are available when as is configured for the Motorola 68000 series.

'-l'

Shorten references to undefined symbols, to one word instead of two.

```
'-m68000 | -m68008 | -m68010 | -m68020 | -m68030'  
' | -m68040 | -m68060 | -m68302 | -m68331 | -m68332'  
' | -m68333 | -m68340 | -mcpu32 | -m5200'
```

Specify what processor in the 68000 family is the target. The default is normally the 68020, but this can be changed at configuration time.

```
'-m68881 | -m68882 | -mno-68881 | -mno-68882'
```

The target machine does (or does not) have a floating-point coprocessor. The default is to assume a coprocessor for 68020, 68030, and cpu32. Although the basic 68000 is not compatible with the 68881, a combination of the two can be specified, since it's possible to do emulation of the coprocessor instructions with the main processor.

```
'-m68851 | -mno-68851'
```

The target machine does (or does not) have a memory-management unit coprocessor. The default is to assume an MMU for 68020 and up.

*Note Nios II Options::, for the options available when as is configured for an Altera Nios II processor.

For details about the PDP-11 machine dependent features options, see *note PDP-11-Options::.

```
'-mpic | -mno-pic'
```

Generate position-independent (or position-dependent) code. The default is '-mpic'.

```
'-mall'
```

```
'-mall-extensions'
```

Enable all instruction set extensions. This is the default.

```
'-mno-extensions'
```

Disable all instruction set extensions.

```
'-mEXTENSION | -mno-EXTENSION'
```

Enable (or disable) a particular instruction set extension.

```
'-mCPU'
```

Enable the instruction set extensions supported by a particular CPU, and disable all other extensions.

```
'-mMACHINE'
```

Enable the instruction set extensions supported by a particular machine model, and disable all other extensions.

The following options are available when as is configured for a picoJava processor.

```
'-mb'
```

Generate "big endian" format output.

```
'-ml'
```

Generate "little endian" format output.

The following options are available when `as` is configured for the Motorola 68HC11 or 68HC12 series.

- '-m68hc11 | -m68hc12 | -m68hcs12 | -mm9s12x | -mm9s12xg'
Specify what processor is the target. The default is defined by the configuration option when building the assembler.
- '--xgate-ramoffset'
Instruct the linker to offset RAM addresses from S12X address space into XGATE address space.
- '-mshort'
Specify to use the 16-bit integer ABI.
- '-mlong'
Specify to use the 32-bit integer ABI.
- '-mshort-double'
Specify to use the 32-bit double ABI.
- '-mlong-double'
Specify to use the 64-bit double ABI.
- '--force-long-branches'
Relative branches are turned into absolute ones. This concerns conditional branches, unconditional branches and branches to a sub routine.
- '-S | --short-branches'
Do not turn relative branches into absolute ones when the offset is out of range.
- '--strict-direct-mode'
Do not turn the direct addressing mode into extended addressing mode when the instruction does not support direct addressing mode.
- '--print-insn-syntax'
Print the syntax of instruction in case of error.
- '--print-opcodes'
Print the list of instructions with syntax and then exit.
- '--generate-example'
Print an example of instruction for each possible instruction and then exit. This option is only useful for testing '`as`'.

The following options are available when '`as`' is configured for the SPARC architecture:

- '-Av6 | -Av7 | -Av8 | -Asparclet | -Asparclite'
- '-Av8plus | -Av8plusa | -Av9 | -Av9a'
Explicitly select a variant of the SPARC architecture.
- '-Av8plus' and '-Av8plusa' select a 32 bit environment. '-Av9' and '-Av9a' select a 64 bit environment.
- '-Av8plusa' and '-Av9a' enable the SPARC V9 instruction set with UltraSPARC extensions.

'-xarch=v8plus | -xarch=v8plusa'

For compatibility with the Solaris v9 assembler. These options are equivalent to -Av8plus and -Av8plusa, respectively.

'-bump'

Warn when the assembler switches to another architecture.

The following options are available when as is configured for the 'c54x architecture.

'-mfar-mode'

Enable extended addressing mode. All addresses and relocations will assume extended addressing (usually 23 bits).

'-mcpu=CPU_VERSION'

Sets the CPU version being compiled for.

'-merrors-to-file FILENAME'

Redirect error output to a file, for broken systems which don't support such behaviour in the shell.

The following options are available when as is configured for a MIPS processor.

'-G NUM'

This option sets the largest size of an object that can be referenced implicitly with the 'gp' register. It is only accepted for targets that use ECOFF format, such as a DECstation running Ultrix. The default value is 8.

'-EB'

Generate "big endian" format output.

'-EL'

Generate "little endian" format output.

'-mips1'

'-mips2'

'-mips3'

'-mips4'

'-mips5'

'-mips32'

'-mips32r2'

'-mips32r3'

'-mips32r5'

'-mips32r6'

'-mips64'

'-mips64r2'

'-mips64r3'

'-mips64r5'

'-mips64r6'

Generate code for a particular MIPS Instruction Set Architecture level. '-mips1' is an alias for '-march=r3000', '-mips2' is an alias for '-march=r6000', '-mips3' is an alias for '-march=r4000' and '-mips4' is an alias for '-march=r8000'. '-mips5', '-mips32', '-mips32r2', '-mips32r3', '-mips32r5', '-mips32r6', '-mips64', '-mips64r2', '-mips64r3', '-mips64r5', and '-mips64r6' correspond to generic MIPS V, MIPS32, MIPS32 Release 2, MIPS32 Release 3, MIPS32 Release 5, MIPS32 Release 6, MIPS64, MIPS64 Release 2, MIPS64 Release 3, MIPS64 Release 5, and MIPS64 Release 6 ISA processors, respectively.

'-march=CPU'
Generate code for a particular MIPS CPU.

'-mtune=CPU'
Schedule and tune for a particular MIPS CPU.

'-mfix7000'
'-mno-fix7000'
Cause nops to be inserted if the read of the destination register of an mfhi or mflo instruction occurs in the following two instructions.

'-mfix-rm7000'
'-mno-fix-rm7000'
Cause nops to be inserted if a dmult or dmultu instruction is followed by a load instruction.

'-mdebug'
'-no-mdebug'
Cause stabs-style debugging output to go into an ECOFF-style .mdebug section instead of the standard ELF .stabs sections.

'-mpdr'
'-mno-pdr'
Control generation of '.pdr' sections.

'-mgp32'
'-mfp32'
The register sizes are normally inferred from the ISA and ABI, but these flags force a certain group of registers to be treated as 32 bits wide at all times. '-mgp32' controls the size of general-purpose registers and '-mfp32' controls the size of floating-point registers.

'-mgp64'
'-mfp64'
The register sizes are normally inferred from the ISA and ABI, but these flags force a certain group of registers to be treated as 64 bits wide at all times. '-mgp64' controls the size of general-purpose registers and '-mfp64' controls the size of floating-point registers.

'-mfpxx'
The register sizes are normally inferred from the ISA and ABI, but using this flag in combination with '-mabi=32' enables an ABI variant which will operate correctly with floating-point registers which are 32 or 64 bits wide.

'-modd-spreg'
'-mno-odd-spreg'
Enable use of floating-point operations on odd-numbered single-precision registers when supported by the ISA. '-mfpxx' implies '-mno-odd-spreg', otherwise the default is '-modd-spreg'.

'-mips16'
'-no-mips16'
Generate code for the MIPS 16 processor. This is equivalent to putting '.set mips16' at the start of the assembly file.

'-no-mips16' turns off this option.

'-mmicromips'

'-mno-micromips'

Generate code for the microMIPS processor. This is equivalent to putting '.set micromips' at the start of the assembly file.

'-mno-micromips' turns off this option. This is equivalent to putting '.set nomicromips' at the start of the assembly file.

'-msmartmips'

'-mno-smartmips'

Enables the SmartMIPS extension to the MIPS32 instruction set.

This is equivalent to putting '.set smartmips' at the start of the assembly file. '-mno-smartmips' turns off this option.

'-mips3d'

'-no-mips3d'

Generate code for the MIPS-3D Application Specific Extension. This tells the assembler to accept MIPS-3D instructions. '-no-mips3d' turns off this option.

'-mdmx'

'-no-mdmx'

Generate code for the MDMX Application Specific Extension. This tells the assembler to accept MDMX instructions. '-no-mdmx' turns off this option.

'-mdsp'

'-mno-dsp'

Generate code for the DSP Release 1 Application Specific Extension. This tells the assembler to accept DSP Release 1 instructions.

'-mno-dsp' turns off this option.

'-mdspr2'

'-mno-dspr2'

Generate code for the DSP Release 2 Application Specific Extension. This option implies '-mdsp'. This tells the assembler to accept DSP Release 2 instructions. '-mno-dspr2' turns off this option.

'-mdspr3'

'-mno-dspr3'

Generate code for the DSP Release 3 Application Specific Extension. This option implies '-mdsp' and '-mdspr2'. This tells the assembler to accept DSP Release 3 instructions. '-mno-dspr3' turns off this option.

'-mmsa'

'-mno-msa'

Generate code for the MIPS SIMD Architecture Extension. This tells the assembler to accept MSA instructions. '-mno-msa' turns off this option.

'-mxpa'

'-mno-xpa'

Generate code for the MIPS eXtended Physical Address (XPA) Extension. This tells the assembler to accept XPA instructions.

'-mno-xpa' turns off this option.

'-mmt'

'-mno-mt'

Generate code for the MT Application Specific Extension. This tells the assembler to accept MT instructions. '-mno-mt' turns off this option.

'-mmcu'

'-mno-mcu'

Generate code for the MCU Application Specific Extension. This tells the assembler to accept MCU instructions. '-mno-mcu' turns off this option.

'-minsn32'

'-mno-insn32'

Only use 32-bit instruction encodings when generating code for the microMIPS processor. This option inhibits the use of any 16-bit instructions. This is equivalent to putting '.set insn32' at the start of the assembly file. '-mno-insn32' turns off this option. This is equivalent to putting '.set noinsn32' at the start of the assembly file. By default '-mno-insn32' is selected, allowing all instructions to be used.

'--construct-floats'

'--no-construct-floats'

The '--no-construct-floats' option disables the construction of double width floating point constants by loading the two halves of the value into the two single width floating point registers that make up the double width register. By default '--construct-floats' is selected, allowing construction of these floating point constants.

'--relax-branch'

'--no-relax-branch'

The '--relax-branch' option enables the relaxation of out-of-range branches. By default '--no-relax-branch' is selected, causing any out-of-range branches to produce an error.

'-mignore-branch-isa'

'-mno-ignore-branch-isa'

Ignore branch checks for invalid transitions between ISA modes. The semantics of branches does not provide for an ISA mode switch, so in most cases the ISA mode a branch has been encoded for has to be the same as the ISA mode of the branch's target label. Therefore GAS has checks implemented that verify in branch assembly that the two ISA modes match. '-mignore-branch-isa' disables these checks. By default '-mno-ignore-branch-isa' is selected, causing any invalid branch requiring a transition between ISA modes to produce an error.

'-mnan=ENCODING'

Select between the IEEE 754-2008 ('-mnan=2008') or the legacy ('-mnan=legacy') NaN encoding format. The latter is the default.

'--emulation=NAME'

This option was formerly used to switch between ELF and ECOFF output on targets like IRIX 5 that supported both. MIPS ECOFF support was removed in GAS 2.24, so the option now serves little purpose. It is retained for backwards compatibility.

The available configuration names are: 'mipsself', 'mipslelf' and

'mipsbelf'. Choosing 'mipself' now has no effect, since the output is always ELF. 'mipslelf' and 'mipsbelf' select little- and big-endian output respectively, but '-EL' and '-EB' are now the preferred options instead.

'-nocpp'

'as' ignores this option. It is accepted for compatibility with the native tools.

'--trap'

'--no-trap'

'--break'

'--no-break'

Control how to deal with multiplication overflow and division by zero. '--trap' or '--no-break' (which are synonyms) take a trap exception (and only work for Instruction Set Architecture level 2 and higher); '--break' or '--no-trap' (also synonyms, and the default) take a break exception.

'-n'

When this option is used, 'as' will issue a warning every time it generates a nop instruction from a macro.

The following options are available when as is configured for an MCore processor.

'-jsri2bsr'

'-nojsri2bsr'

Enable or disable the JSRI to BSR transformation. By default this is enabled. The command line option '-nojsri2bsr' can be used to disable it.

'-sifilter'

'-nosifilter'

Enable or disable the silicon filter behaviour. By default this is disabled. The default can be overridden by the '-sifilter' command line option.

'-relax'

Alter jump instructions for long displacements.

'-mcpu=[210|340]'

Select the cpu type on the target hardware. This controls which instructions can be assembled.

'-EB'

Assemble for a big endian target.

'-EL'

Assemble for a little endian target.

*Note Meta Options::, for the options available when as is configured for a Meta processor.

See the info pages for documentation of the MMIX-specific options.

*Note NDS32 Options::, for the options available when as is configured for a NDS32 processor.

*Note PowerPC-Opts::, for the options available when as is configured for a PowerPC processor.

*Note RISC-V-Opts::, for the options available when as is configured for a RISC-V processor.

See the info pages for documentation of the RX-specific options.

The following options are available when as is configured for the s390 processor family.

```
'-m31'
'-m64'
    Select the word size, either 31/32 bits or 64 bits.
'-mesa'
'-mzarch'
    Select the architecture mode, either the Enterprise System
    Architecture (esa) or the z/Architecture mode (zarch).
'-march=PROCESSOR'
    Specify which s390 processor variant is the target, 'g5' (or
    'arch3'), 'g6', 'z900' (or 'arch5'), 'z990' (or 'arch6'), 'z9-109',
    'z9-ec' (or 'arch7'), 'z10' (or 'arch8'), 'z196' (or 'arch9'),
    'zEC12' (or 'arch10'), 'z13' (or 'arch11'), or 'arch12'.
'-mregnames'
'-mno-regnames'
    Allow or disallow symbolic names for registers.
'-mwarn-areg-zero'
    Warn whenever the operand for a base or index register has been
    specified but evaluates to zero.
```

*Note TIC6X Options::, for the options available when as is configured for a TMS320C6000 processor.

*Note TILE-Gx Options::, for the options available when as is configured for a TILE-Gx processor.

*Note Visium Options::, for the options available when as is configured for a Visium processor.

*Note Xtensa Options::, for the options available when as is configured for an Xtensa processor.

The following options are available when as is configured for a Z80 family processor.

```
'-z80'
    Assemble for Z80 processor.
'-r800'
    Assemble for R800 processor.
'-ignore-undocumented-instructions'
'-Wnud'
    Assemble undocumented Z80 instructions that also work on R800
    without warning.
'-ignore-unportable-instructions'
'-Wnup'
    Assemble all undocumented Z80 instructions without warning.
'-warn-undocumented-instructions'
'-Wud'
    Issue a warning for undocumented Z80 instructions that also work on
    R800.
```

```
'-warn-unportable-instructions'
'-Wup'
    Issue a warning for undocumented Z80 instructions that do not work
    on R800.
'-forbid-undocumented-instructions'
'-Fud'
    Treat all undocumented instructions as errors.
'-forbid-unportable-instructions'
'-Fup'
    Treat undocumented Z80 instructions that do not work on R800 as
    errors.
```

* Menu:

```
* Manual::          Structure of this Manual
* GNU Assembler::  The GNU Assembler
* Object Formats::  Object File Formats
* Command Line::   Command Line
* Input Files::     Input Files
* Object::          Output (Object) File
* Errors::          Error and Warning Messages
```



File: as.info, Node: Manual, Next: GNU Assembler, Up: Overview

1.1 Structure of this Manual

=====

This manual is intended to describe what you need to know to use GNU 'as'. We cover the syntax expected in source files, including notation for symbols, constants, and expressions; the directives that 'as' understands; and of course how to invoke 'as'.

This manual also describes some of the machine-dependent features of various flavors of the assembler.

On the other hand, this manual is not intended as an introduction to programming in assembly language--let alone programming in general! In a similar vein, we make no attempt to introduce the machine architecture; we do not describe the instruction set, standard mnemonics, registers or addressing modes that are standard to a particular architecture. You may want to consult the manufacturer's machine architecture manual for this information.



File: as.info, Node: GNU Assembler, Next: Object Formats, Prev: Manual, Up: Overview

1.2 The GNU Assembler

=====

GNU 'as' is really a family of assemblers. If you use (or have used) the GNU assembler on one architecture, you should find a fairly similar environment when you use it on another architecture. Each version has much in common with the others, including object file formats, most assembler directives (often called "pseudo-ops") and assembler syntax.

'as' is primarily intended to assemble the output of the GNU C compiler 'gcc' for use by the linker 'ld'. Nevertheless, we've tried to

make 'as' assemble correctly everything that other assemblers for the same machine would assemble. Any exceptions are documented explicitly (*note Machine Dependencies:). This doesn't mean 'as' always uses the same syntax as another assembler for the same architecture; for example, we know of several incompatible versions of 680x0 assembly language syntax.

Unlike older assemblers, 'as' is designed to assemble a source program in one pass of the source file. This has a subtle impact on the '.org' directive (*note '.org': Org.).



File: as.info, Node: Object Formats, Next: Command Line, Prev: GNU Assembler, Up: Overview

1.3 Object File Formats

=====

The GNU assembler can be configured to produce several alternative object file formats. For the most part, this does not affect how you write assembly language programs; but directives for debugging symbols are typically different in different file formats. *Note Symbol Attributes: Symbol Attributes.



File: as.info, Node: Command Line, Next: Input Files, Prev: Object Formats, Up: Overview

1.4 Command Line

=====

After the program name 'as', the command line may contain options and file names. Options may appear in any order, and may be before, after, or between file names. The order of file names is significant.

'--' (two hyphens) by itself names the standard input file explicitly, as one of the files for 'as' to assemble.

Except for '--' any command line argument that begins with a hyphen ('-') is an option. Each option changes the behavior of 'as'. No option changes the way another option works. An option is a '-' followed by one or more letters; the case of the letter is important. All options are optional.

Some options expect exactly one file name to follow them. The file name may either immediately follow the option's letter (compatible with older assemblers) or it may be the next command argument (GNU standard). These two command lines are equivalent:

```
as -o my-object-file.o mumble.s
as -omy-object-file.o mumble.s
```



File: as.info, Node: Input Files, Next: Object, Prev: Command Line, Up: Overview

1.5 Input Files

=====

We use the phrase "source program", abbreviated "source", to describe

the program input to one run of 'as'. The program may be in one or more files; how the source is partitioned into files doesn't change the meaning of the source.

The source program is a concatenation of the text in all the files, in the order specified.

Each time you run 'as' it assembles exactly one source program. The source program is made up of one or more files. (The standard input is also a file.)

You give 'as' a command line that has zero or more input file names. The input files are read (from left file name to right). A command line argument (in any position) that has no special meaning is taken to be an input file name.

If you give 'as' no file names it attempts to read one input file from the 'as' standard input, which is normally your terminal. You may have to type <ctl-D> to tell 'as' there is no more program to assemble.

Use '--' if you need to explicitly name the standard input file in your command line.

If the source is empty, 'as' produces a small, empty object file.

Filename and Line-numbers

There are two ways of locating a line in the input file (or files) and either may be used in reporting error messages. One way refers to a line number in a physical file; the other refers to a line number in a "logical" file. *Note Error and Warning Messages: Errors.

"Physical files" are those files named in the command line given to 'as'.

"Logical files" are simply names declared explicitly by assembler directives; they bear no relation to physical files. Logical file names help error messages reflect the original source file, when 'as' source is itself synthesized from other files. 'as' understands the '#' directives emitted by the 'gcc' preprocessor. See also *note '.file': File.



File: as.info, Node: Object, Next: Errors, Prev: Input Files, Up: Overview

1.6 Output (Object) File

=====

Every time you run 'as' it produces an output file, which is your assembly language program translated into numbers. This file is the object file. Its default name is 'a.out'. You can give it another name by using the '-o' option. Conventionally, object file names end with '.o'. The default name is used for historical reasons: older assemblers were capable of assembling self-contained programs directly into a runnable program. (For some formats, this isn't currently possible, but it can be done for the 'a.out' format.)

The object file is meant for input to the linker 'ld'. It contains

assembled program code, information to help 'ld' integrate the assembled program into a runnable file, and (optionally) symbolic information for the debugger.



File: as.info, Node: Errors, Prev: Object, Up: Overview

1.7 Error and Warning Messages

=====

'as' may write warnings and error messages to the standard error file (usually your terminal). This should not happen when a compiler runs 'as' automatically. Warnings report an assumption made so that 'as' could keep assembling a flawed program; errors report a grave problem that stops the assembly.

Warning messages have the format

```
file_name:NNN:Warning Message Text
```

(where NNN is a line number). If both a logical file name (*note '.file': File.) and a logical line number (*note '.line': Line.) have been given then they will be used, otherwise the file name and line number in the current assembler source file will be used. The message text is intended to be self explanatory (in the grand Unix tradition).

Note the file name must be set via the logical version of the '.file' directive, not the DWARF2 version of the '.file' directive. For example:

```
.file 2 "bar.c"
  error_assembler_source
.file "foo.c"
.line 30
  error_c_source
```

produces this output:

```
Assembler messages:
asm.s:2: Error: no such instruction: `error_assembler_source'
foo.c:31: Error: no such instruction: `error_c_source'
```

Error messages have the format

```
file_name:NNN:FATAL>Error Message Text
```

The file name and line number are derived as for warning messages. The actual message text may be rather less explanatory because many of them aren't supposed to happen.



File: as.info, Node: Invoking, Next: Syntax, Prev: Overview, Up: Top

2 Command-Line Options

This chapter describes command-line options available in all versions of the GNU assembler; see *note Machine Dependencies::, for options specific to particular machine architectures.

If you are invoking 'as' via the GNU C compiler, you can use the '-Wa' option to pass arguments through to the assembler. The assembler arguments must be separated from each other (and the '-Wa') by commas. For example:

```
gcc -c -g -O -Wa,-alh,-L file.c
```

This passes two options to the assembler: '-alh' (emit a listing to standard output with high-level and assembly source) and '-L' (retain local symbols in the symbol table).

Usually you do not need to use this '-Wa' mechanism, since many compiler command-line options are automatically passed to the assembler by the compiler. (You can call the GNU compiler driver with the '-v' option to see precisely what options it passes to each compilation pass, including the assembler.)

* Menu:

```
* a::          -a[cdghlns] enable listings
* alternate::  -alternate enable alternate macro syntax
* D::         -D for compatibility
* f::         -f to work faster
* I::         -I for .include search path
* K::         -K for difference tables

* L::         -L to retain local symbols
* listing::   -listing-XXX to configure listing output
* M::         -M or -mri to assemble in MRI compatibility mode
* MD::        -MD for dependency tracking
* no-pad-sections:: -no-pad-sections to stop section padding
* o::         -o to name the object file
* R::         -R to join data and text sections
* statistics:: -statistics to see statistics about assembly
* traditional-format:: -traditional-format for compatible output
* v::         -v to announce version
* W::         -W, -no-warn, -warn, -fatal-warnings to control warnings
* Z::         -Z to make object file even after errors
```



File: as.info, Node: a, Next: alternate, Up: Invoking

2.1 Enable Listings: '-a[cdghlns]'

=====

These options enable listing output from the assembler. By itself, '-a' requests high-level, assembly, and symbols listing. You can use other letters to select specific options for the list: '-ah' requests a high-level language listing, '-al' requests an output-program assembly listing, and '-as' requests a symbol table listing. High-level listings require that a compiler debugging option like '-g' be used, and that assembly listings ('-al') be requested also.

Use the '-ag' option to print a first section with general assembly information, like as version, switches passed, or time stamp.

Use the '-ac' option to omit false conditionals from a listing. Any lines which are not assembled because of a false '.if' (or '.ifdef', or

any other conditional), or a true `.if` followed by an `.else`, will be omitted from the listing.

Use the `-ad` option to omit debugging directives from the listing.

Once you have specified one of these options, you can further control listing output and its appearance using the directives `.list`, `.nolist`, `.psize`, `.eject`, `.title`, and `.sbttl`. The `-an` option turns off all forms processing. If you do not request listing output with one of the `-a` options, the listing-control directives have no effect.

The letters after `-a` may be combined into one option, *e.g.*, `-aln`.

Note if the assembler source is coming from the standard input (e.g., because it is being created by `gcc` and the `-pipe` command line switch is being used) then the listing will not contain any comments or preprocessor directives. This is because the listing code buffers input source lines from `stdin` only after they have been preprocessed by the assembler. This reduces memory usage and makes the code more efficient.

US

File: as.info, Node: alternate, Next: D, Prev: a, Up: Invoking

2.2 '--alternate'

=====

Begin in alternate macro mode, see **note* `.altmacro`: Altmacro.

US

File: as.info, Node: D, Next: f, Prev: alternate, Up: Invoking

2.3 '-D'

=====

This option has no effect whatsoever, but it is accepted to make it more likely that scripts written for other assemblers also work with `as`.

US

File: as.info, Node: f, Next: I, Prev: D, Up: Invoking

2.4 Work Faster: '-f'

=====

`-f` should only be used when assembling programs written by a (trusted) compiler. `-f` stops the assembler from doing whitespace and comment preprocessing on the input file(s) before assembling them. **Note* Preprocessing: Preprocessing.

Warning: if you use `-f` when the files actually need to be preprocessed (if they contain comments, for example), `as` does not work correctly.

US

File: as.info, Node: I, Next: K, Prev: f, Up: Invoking

2.5 '.include' Search Path: '-I' PATH

=====

Use this option to add a PATH to the list of directories 'as' searches for files specified in '.include' directives (*note '.include': Include.). You may use '-I' as many times as necessary to include a variety of paths. The current working directory is always searched first; after that, 'as' searches any '-I' directories in the same order as they were specified (left to right) on the command line.



File: as.info, Node: K, Next: L, Prev: I, Up: Invoking

2.6 Difference Tables: '-K'

=====

'as' sometimes alters the code emitted for directives of the form '.word SYM1-SYM2'. *Note '.word': Word. You can use the '-K' option if you want a warning issued when this is done.



File: as.info, Node: L, Next: listing, Prev: K, Up: Invoking

2.7 Include Local Symbols: '-L'

=====

Symbols beginning with system-specific local label prefixes, typically '.L' for ELF systems or 'L' for traditional a.out systems, are called "local symbols". *Note Symbol Names::. Normally you do not see such symbols when debugging, because they are intended for the use of programs (like compilers) that compose assembler programs, not for your notice. Normally both 'as' and 'ld' discard such symbols, so you do not normally debug with them.

This option tells 'as' to retain those local symbols in the object file. Usually if you do this you also tell the linker 'ld' to preserve those symbols.



File: as.info, Node: listing, Next: M, Prev: L, Up: Invoking

2.8 Configuring listing output: '--listing'

=====

The listing feature of the assembler can be enabled via the command line switch '-a' (*note a:). This feature combines the input source file(s) with a hex dump of the corresponding locations in the output object file, and displays them as a listing file. The format of this listing can be controlled by directives inside the assembler source (i.e., '.list' (*note List:), '.title' (*note Title:), '.sbttl' (*note Sbttl:), '.psize' (*note Psize:), and '.eject' (*note Eject:)) and also by the following switches:

'--listing-lhs-width='number''

Sets the maximum width, in words, of the first line of the hex byte dump. This dump appears on the left hand side of the listing output.

'--listing-lhs-width2='number''

Sets the maximum width, in words, of any further lines of the hex byte dump for a given input source line. If this value is not

specified, it defaults to being the same as the value specified for '--listing-lhs-width'. If neither switch is used the default is to one.

'--listing-rhs-width='number''

Sets the maximum width, in characters, of the source line that is displayed alongside the hex dump. The default value for this parameter is 100. The source line is displayed on the right hand side of the listing output.

'--listing-cont-lines='number''

Sets the maximum number of continuation lines of hex dump that will be displayed for a given single line of source input. The default value is 4.



File: as.info, Node: M, Next: MD, Prev: listing, Up: Invoking

2.9 Assemble in MRI Compatibility Mode: '-M'

=====

The '-M' or '--mri' option selects MRI compatibility mode. This changes the syntax and pseudo-op handling of 'as' to make it compatible with the 'ASM68K' or the 'ASM960' (depending upon the configured target) assembler from Microtec Research. The exact nature of the MRI syntax will not be documented here; see the MRI manuals for more information. Note in particular that the handling of macros and macro arguments is somewhat different. The purpose of this option is to permit assembling existing MRI assembler code using 'as'.

The MRI compatibility is not complete. Certain operations of the MRI assembler depend upon its object file format, and can not be supported using other object file formats. Supporting these would require enhancing each object file format individually. These are:

* global symbols in common section

The m68k MRI assembler supports common sections which are merged by the linker. Other object file formats do not support this. 'as' handles common sections by treating them as a single common symbol. It permits local symbols to be defined within a common section, but it can not support global symbols, since it has no way to describe them.

* complex relocations

The MRI assemblers support relocations against a negated section address, and relocations which combine the start addresses of two or more sections. These are not support by other object file formats.

* 'END' pseudo-op specifying start address

The MRI 'END' pseudo-op permits the specification of a start address. This is not supported by other object file formats. The start address may instead be specified using the '-e' option to the linker, or in a linker script.

* 'IDNT', '.ident' and 'NAME' pseudo-ops

The MRI 'IDNT', '.ident' and 'NAME' pseudo-ops assign a module name to the output file. This is not supported by other object file formats.

* 'ORG' pseudo-op

The m68k MRI 'ORG' pseudo-op begins an absolute section at a given address. This differs from the usual 'as' '.org' pseudo-op, which changes the location within the current section. Absolute sections are not supported by other object file formats. The address of a section may be assigned within a linker script.

There are some other features of the MRI assembler which are not supported by 'as', typically either because they are difficult or because they seem of little consequence. Some of these may be supported in future releases.

* EBCDIC strings

EBCDIC strings are not supported.

* packed binary coded decimal

Packed binary coded decimal is not supported. This means that the 'DC.P' and 'DCB.P' pseudo-ops are not supported.

* 'FEQU' pseudo-op

The m68k 'FEQU' pseudo-op is not supported.

* 'NOOBJ' pseudo-op

The m68k 'NOOBJ' pseudo-op is not supported.

* 'OPT' branch control options

The m68k 'OPT' branch control options--'B', 'BRS', 'BRB', 'BRL', and 'BRW'--are ignored. 'as' automatically relaxes all branches, whether forward or backward, to an appropriate size, so these options serve no purpose.

* 'OPT' list control options

The following m68k 'OPT' list control options are ignored: 'C', 'CEX', 'CL', 'CRE', 'E', 'G', 'I', 'M', 'MEX', 'MC', 'MD', 'X'.

* other 'OPT' options

The following m68k 'OPT' options are ignored: 'NEST', 'O', 'OLD', 'OP', 'P', 'PCO', 'PCR', 'PCS', 'R'.

* 'OPT' 'D' option is default

The m68k 'OPT' 'D' option is the default, unlike the MRI assembler. 'OPT NOD' may be used to turn it off.

* 'XREF' pseudo-op.

The m68k 'XREF' pseudo-op is ignored.

* '.debug' pseudo-op

The i960 '.debug' pseudo-op is not supported.

* '.extended' pseudo-op

The i960 '.extended' pseudo-op is not supported.

* '.list' pseudo-op.

The various options of the i960 '.list' pseudo-op are not supported.

* '.optimize' pseudo-op

The i960 '.optimize' pseudo-op is not supported.

* '.output' pseudo-op

The i960 '.output' pseudo-op is not supported.

* '.setreal' pseudo-op

The i960 '.setreal' pseudo-op is not supported.

US

File: as.info, Node: MD, Next: no-pad-sections, Prev: M, Up: Invoking

2.10 Dependency Tracking: '--MD'

=====

'as' can generate a dependency file for the file it creates. This file consists of a single rule suitable for 'make' describing the dependencies of the main source file.

The rule is written to the file named in its argument.

This feature is used in the automatic updating of makefiles.

US

File: as.info, Node: no-pad-sections, Next: o, Prev: MD, Up: Invoking

2.11 Output Section Padding

=====

Normally the assembler will pad the end of each output section up to its alignment boundary. But this can waste space, which can be significant on memory constrained targets. So the '--no-pad-sections' option will disable this behaviour.

US

File: as.info, Node: o, Next: R, Prev: no-pad-sections, Up: Invoking

2.12 Name the Object File: '-o'

=====

There is always one object file output when you run 'as'. By default it

has the name 'a.out' (or 'b.out', for Intel 960 targets only). You use this option (which takes exactly one filename) to give the object file a different name.

Whatever the object file is called, 'as' overwrites any existing file of the same name.



File: as.info, Node: R, Next: statistics, Prev: o, Up: Invoking

2.13 Join Data and Text Sections: '-R'

=====

'-R' tells 'as' to write the object file as if all data-section data lives in the text section. This is only done at the very last moment: your binary data are the same, but data section parts are relocated differently. The data section part of your object file is zero bytes long because all its bytes are appended to the text section. (*Note Sections and Relocation: Sections.)

When you specify '-R' it would be possible to generate shorter address displacements (because we do not have to cross between text and data section). We refrain from doing this simply for compatibility with older versions of 'as'. In future, '-R' may work this way.

When 'as' is configured for COFF or ELF output, this option is only useful if you use sections named '.text' and '.data'.

'-R' is not supported for any of the HPPA targets. Using '-R' generates a warning from 'as'.



File: as.info, Node: statistics, Next: traditional-format, Prev: R, Up: Invoking

2.14 Display Assembly Statistics: '--statistics'

=====

Use '--statistics' to display two statistics about the resources used by 'as': the maximum amount of space allocated during the assembly (in bytes), and the total execution time taken for the assembly (in CPU seconds).



File: as.info, Node: traditional-format, Next: v, Prev: statistics, Up: Invoking

2.15 Compatible Output: '--traditional-format'

=====

For some targets, the output of 'as' is different in some ways from the output of some existing assembler. This switch requests 'as' to use the traditional format instead.

For example, it disables the exception frame optimizations which 'as' normally does by default on 'gcc' output.



File: as.info, Node: v, Next: W, Prev: traditional-format, Up: Invoking

2.16 Announce Version: '-v'

=====

You can find out what version of as is running by including the option '-v' (which you can also spell as '-version') on the command line.

US

File: as.info, Node: W, Next: Z, Prev: v, Up: Invoking

2.17 Control Warnings: '-W', '--warn', '--no-warn', '--fatal-warnings'

=====

'as' should never give a warning or error message when assembling compiler output. But programs written by people often cause 'as' to give a warning that a particular assumption was made. All such warnings are directed to the standard error file.

If you use the '-W' and '--no-warn' options, no warnings are issued. This only affects the warning messages: it does not change any particular of how 'as' assembles your file. Errors, which stop the assembly, are still reported.

If you use the '--fatal-warnings' option, 'as' considers files that generate warnings to be in error.

You can switch these options off again by specifying '--warn', which causes warnings to be output as usual.

US

File: as.info, Node: Z, Prev: W, Up: Invoking

2.18 Generate Object File in Spite of Errors: '-Z'

=====

After an error message, 'as' normally produces no output. If for some reason you are interested in object file output even after 'as' gives an error message on your program, use the '-Z' option. If there are any errors, 'as' continues anyways, and writes an object file after a final warning message of the form 'N errors, M warnings, generating bad object file.'

US

File: as.info, Node: Syntax, Next: Sections, Prev: Invoking, Up: Top

3 Syntax

This chapter describes the machine-independent syntax allowed in a source file. 'as' syntax is similar to what many other assemblers use; it is inspired by the BSD 4.2 assembler, except that 'as' does not assemble Vax bit-fields.

* Menu:

* Preprocessing::	Preprocessing
* Whitespace::	Whitespace
* Comments::	Comments
* Symbol Intro::	Symbols
* Statements::	Statements
* Constants::	Constants



File: as.info, Node: Preprocessing, Next: Whitespace, Up: Syntax

3.1 Preprocessing

=====

The 'as' internal preprocessor:

- * adjusts and removes extra whitespace. It leaves one space or tab before the keywords on a line, and turns any other whitespace on the line into a single space.
- * removes all comments, replacing them with a single space, or an appropriate number of newlines.
- * converts character constants into the appropriate numeric values.

It does not do macro processing, include file handling, or anything else you may get from your C compiler's preprocessor. You can do include file processing with the '.include' directive (*note '.include': Include.). You can use the GNU C compiler driver to get other "CPP" style preprocessing by giving the input file a '.S' suffix. *Note Options Controlling the Kind of Output: (gcc info)Overall Options.

Excess whitespace, comments, and character constants cannot be used in the portions of the input text that are not preprocessed.

If the first line of an input file is '#NO_APP' or if you use the '-f' option, whitespace and comments are not removed from the input file. Within an input file, you can ask for whitespace and comment removal in specific portions of the by putting a line that says '#APP' before the text that may contain whitespace or comments, and putting a line that says '#NO_APP' after this text. This feature is mainly intend to support 'asm' statements in compilers whose output is otherwise free of comments and whitespace.



File: as.info, Node: Whitespace, Next: Comments, Prev: Preprocessing, Up: Syntax

3.2 Whitespace

=====

"Whitespace" is one or more blanks or tabs, in any order. Whitespace is used to separate symbols, and to make programs neater for people to read. Unless within character constants (*note Character Constants: Characters.), any whitespace means the same as exactly one space.



File: as.info, Node: Comments, Next: Symbol Intro, Prev: Whitespace, Up: Syntax

3.3 Comments

=====

There are two ways of rendering comments to 'as'. In both cases the comment is equivalent to one space.

Anything from '/*' through the next '*/' is a comment. This means you may not nest these comments.

```

/*
  The only way to include a newline ('\n') in a comment
  is to use this sort of comment.
*/

/* This sort of comment does not nest. */

```

Anything from a "line comment" character up to the next newline is considered a comment and is ignored. The line comment character is target specific, and some targets multiple comment characters. Some targets also have line comment characters that only work if they are the first character on a line. Some targets use a sequence of two characters to introduce a line comment. Some targets can also change their line comment characters depending upon command line options that have been used. For more details see the `_Syntax_` section in the documentation for individual targets.

If the line comment character is the hash sign ('#') then it still has the special ability to enable and disable preprocessing (`*note Preprocessing:*`) and to specify logical line numbers:

To be compatible with past assemblers, lines that begin with '#' have a special interpretation. Following the '#' should be an absolute expression (`*note Expressions:*`): the logical line number of the `_next_line`. Then a string (`*note Strings: Strings.*`) is allowed: if present it is a new logical file name. The rest of the line, if any, should be whitespace.

If the first non-whitespace characters on the line are not numeric, the line is ignored. (Just like a comment.)

```

# This is an ordinary comment.
# 42-6 "new_file_name" # New logical file name
# This is logical line # 36.

```

This feature is deprecated, and may disappear from future versions of 'as'.



File: as.info, Node: Symbol Intro, Next: Statements, Prev: Comments, Up: Syntax

3.4 Symbols

=====

A "symbol" is one or more characters chosen from the set of all letters (both upper and lower case), digits and the three characters `_. $`. On most machines, you can also use `'$'` in symbol names; exceptions are noted in `*note Machine Dependencies:*`. No symbol may begin with a digit. Case is significant. There is no length limit; all characters are significant. Multibyte characters are supported. Symbols are delimited by characters not in that set, or by the beginning of a file (since the source program must end with a newline, the end of a file is not a possible symbol delimiter). `*Note Symbols:*`.

Symbol names may also be enclosed in double quote `''` characters. In such cases any characters are allowed, except for the NUL character. If a double quote character is to be included in the symbol name it must be preceded by a backslash `'\'` character.



File: as.info, Node: Statements, Next: Constants, Prev: Symbol Intro, Up: Syntax

3.5 Statements

=====

A "statement" ends at a newline character ('\n') or a "line separator character". The line separator character is target specific and described in the `_Syntax_` section of each target's documentation. Not all targets support a line separator character. The newline or line separator character is considered to be part of the preceding statement. Newlines and separators within character constants are an exception: they do not end statements.

It is an error to end any statement with end-of-file: the last character of any input file should be a newline.

An empty statement is allowed, and may include whitespace. It is ignored.

A statement begins with zero or more labels, optionally followed by a key symbol which determines what kind of statement it is. The key symbol determines the syntax of the rest of the statement. If the symbol begins with a dot '.' then the statement is an assembler directive: typically valid for any computer. If the symbol begins with a letter the statement is an assembly language "instruction": it assembles into a machine language instruction. Different versions of 'as' for different computers recognize different instructions. In fact, the same symbol may represent a different instruction in a different computer's assembly language.

A label is a symbol immediately followed by a colon (':'). Whitespace before a label or after a colon is permitted, but you may not have whitespace between a label's symbol and its colon. *Note Labels::.

For HPPA targets, labels need not be immediately followed by a colon, but the definition of a label must begin in column zero. This also implies that only one label may be defined on each line.

```
label:      .directive      followed by something
another_label:      # This is an empty statement.
                instruction  operand_1, operand_2, ...
```



File: as.info, Node: Constants, Prev: Statements, Up: Syntax

3.6 Constants

=====

A constant is a number, written so that its value is known by inspection, without knowing any context. Like this:

```
.byte  74, 0112, 092, 0x4A, 0X4a, 'J', '\J # All the same value.
.ascii "Ring the bell\7"                    # A string constant.
.octa  0x123456789abcdef0123456789ABCDEF0 # A bignum.
.float 0f-314159265358979323846264338327\
95028841971.693993751E-40                    # - pi, a flonum.
```

* Menu:

* Characters:: Character Constants

`'\\'`

Represents one `'\'` character.

`'\"'`

Represents one `'\"'` character. Needed in strings to represent this character, because an unescaped `'\"'` would end the string.

`'\ ANYTHING-ELSE'`

Any other character when escaped by `'\'` gives a warning, but assembles as if the `'\'` was not present. The idea is that if you used an escape sequence you clearly didn't want the literal interpretation of the following character. However `'as'` has no other interpretation, so `'as'` knows it is giving you the wrong code and warns you of the fact.

Which characters are escapable, and what those escapes represent, varies widely among assemblers. The current set is what we think the BSD 4.2 assembler recognizes, and is a subset of what most C compilers recognize. If you are in doubt, do not use an escape sequence.

`US`

File: as.info, Node: Chars, Prev: Strings, Up: Characters

3.6.1.2 Characters

.....

A single character may be written as a single quote immediately followed by that character. Some backslash escapes apply to characters, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, and `'\"'` with the same meaning as for strings, plus `'\''` for a single quote. So if you want to write the character backslash, you must write `'\\'` where the first `'\'` escapes the second `'\'`. As you can see, the quote is an acute accent, not a grave accent. A newline immediately following an acute accent is taken as a literal character and does not count as the end of a statement. The value of a character constant in a numeric expression is the machine's byte-wide code for that character. `'as'` assumes your character code is ASCII: `'A'` means 65, `'B'` means 66, and so on.

`US`

File: as.info, Node: Numbers, Prev: Characters, Up: Constants

3.6.2 Number Constants

`'as'` distinguishes three kinds of numbers according to how they are stored in the target machine. `_Integers_` are numbers that would fit into an `'int'` in the C language. `_Bignums_` are integers, but they are stored in more than 32 bits. `_Flonums_` are floating point numbers, described below.

* Menu:

* Integers::	Integers
* Bignums::	Bignums
* Flonums::	Flonums

`US`

File: as.info, Node: Integers, Next: Bignums, Up: Numbers

3.6.2.1 Integers

.....

A binary integer is '0b' or '0B' followed by zero or more of the binary digits '01'.

An octal integer is '0' followed by zero or more of the octal digits ('01234567').

A decimal integer starts with a non-zero digit followed by zero or more digits ('0123456789').

A hexadecimal integer is '0x' or '0X' followed by one or more hexadecimal digits chosen from '0123456789abcdefABCDEF'.

Integers have the usual values. To denote a negative integer, use the prefix operator '-' discussed under expressions (*note Prefix Operators: Prefix Ops.).



File: as.info, Node: Bignums, Next: Flonums, Prev: Integers, Up: Numbers

3.6.2.2 Bignums

.....

A "bignum" has the same syntax and semantics as an integer except that the number (or its negative) takes more than 32 bits to represent in binary. The distinction is made because in some places integers are permitted while bignums are not.



File: as.info, Node: Flonums, Prev: Bignums, Up: Numbers

3.6.2.3 Flonums

.....

A "flonum" represents a floating point number. The translation is indirect: a decimal floating point number from the text is converted by 'as' to a generic binary floating point number of more than sufficient precision. This generic floating point number is converted to a particular computer's floating point format (or formats) by a portion of 'as' specialized to that computer.

A flonum is written by writing (in order)

* The digit '0'. ('0' is optional on the HPPA.)

* A letter, to tell 'as' the rest of the number is a flonum. 'e' is recommended. Case is not important.

On the H8/300, Renesas / SuperH SH, and AMD 29K architectures, the letter must be one of the letters 'DFPRSX' (in upper or lower case).

On the ARC, the letter must be one of the letters 'DFRS' (in upper or lower case).

On the Intel 960 architecture, the letter must be one of the letters 'DFT' (in upper or lower case).

On the HPPA architecture, the letter must be 'E' (upper case only).

- * An optional sign: either '+' or '-'.
- * An optional "integer part": zero or more decimal digits.
- * An optional "fractional part": '.' followed by zero or more decimal digits.
- * An optional exponent, consisting of:
 - * An 'E' or 'e'.
 - * Optional sign: either '+' or '-'.
 - * One or more decimal digits.

At least one of the integer part or the fractional part must be present. The floating point number has the usual base-10 value.

'as' does all processing using integers. Flonums are computed independently of any floating point hardware in the computer running 'as'.



File: as.info, Node: Sections, Next: Symbols, Prev: Syntax, Up: Top

4 Sections and Relocation

* Menu:

* Secs Background::	Background
* Ld Sections::	Linker Sections
* As Sections::	Assembler Internal Sections
* Sub-Sections::	Sub-Sections
* bss::	bss Section



File: as.info, Node: Secs Background, Next: Ld Sections, Up: Sections

4.1 Background

=====

Roughly, a section is a range of addresses, with no gaps; all data "in" those addresses is treated the same for some particular purpose. For example there may be a "read only" section.

The linker 'ld' reads many object files (partial programs) and combines their contents to form a runnable program. When 'as' emits an object file, the partial program is assumed to start at address 0. 'ld' assigns the final addresses for the partial program, so that different partial programs do not overlap. This is actually an oversimplification, but it suffices to explain how 'as' uses sections.

'ld' moves blocks of bytes of your program to their run-time addresses. These blocks slide to their run-time addresses as rigid units; their length does not change and neither does the order of bytes within them. Such a rigid unit is called a `_section_`. Assigning run-time addresses to sections is called "relocation". It includes the task of adjusting mentions of object-file addresses so they refer to the

proper run-time addresses. For the H8/300, and for the Renesas / SuperH SH, 'as' pads sections if needed to ensure they end on a word (sixteen bit) boundary.

An object file written by 'as' has at least three sections, any of which may be empty. These are named "text", "data" and "bss" sections.

When it generates COFF or ELF output, 'as' can also generate whatever other named sections you specify using the '.section' directive (*note '.section': Section.). If you do not use any directives that place output in the '.text' or '.data' sections, these sections still exist, but are empty.

When 'as' generates SOM or ELF output for the HPPA, 'as' can also generate whatever other named sections you specify using the '.space' and '.subspace' directives. See 'HP9000 Series 800 Assembly Language Reference Manual' (HP 92432-90001) for details on the '.space' and '.subspace' assembler directives.

Additionally, 'as' uses different names for the standard text, data, and bss sections when generating SOM output. Program text is placed into the '\$CODE\$' section, data into '\$DATA\$', and BSS into '\$BSS\$'.

Within the object file, the text section starts at address '0', the data section follows, and the bss section follows the data section.

When generating either SOM or ELF output files on the HPPA, the text section starts at address '0', the data section at address '0x4000000', and the bss section follows the data section.

To let 'ld' know which data changes when the sections are relocated, and how to change that data, 'as' also writes to the object file details of the relocation needed. To perform relocation 'ld' must know, each time an address in the object file is mentioned:

- * Where in the object file is the beginning of this reference to an address?
- * How long (in bytes) is this reference?
- * Which section does the address refer to? What is the numeric value of
(ADDRESS) - (START-ADDRESS OF SECTION)?
- * Is the reference to an address "Program-Counter relative"?

In fact, every address 'as' ever uses is expressed as
(SECTION) + (OFFSET INTO SECTION)

Further, most expressions 'as' computes have this section-relative nature. (For some object formats, such as SOM for the HPPA, some expressions are symbol-relative instead.)

In this manual we use the notation {SECNAME N} to mean "offset N into section SECNAME."

Apart from text, data and bss sections you need to know about the "absolute" section. When 'ld' mixes partial programs, addresses in the absolute section remain unchanged. For example, address '{absolute 0}' is "relocated" to run-time address 0 by 'ld'. Although the linker never arranges two partial programs' data sections with overlapping addresses after linking, by definition their absolute sections must overlap. Address '{absolute 239}' in one part of a program is always the same address when the program is running as address '{absolute 239}' in any

other part of the program.

The idea of sections is extended to the "undefined" section. Any address whose section is unknown at assembly time is by definition rendered {undefined U}--where U is filled in later. Since numbers are always defined, the only way to generate an undefined address is to mention an undefined symbol. A reference to a named common block would be such a symbol: its value is unknown at assembly time so it has section `_undefined_`.

By analogy the word `_section_` is used to describe groups of sections in the linked program. 'ld' puts all partial programs' text sections in contiguous addresses in the linked program. It is customary to refer to the `_text section_` of a program, meaning all the addresses of all partial programs' text sections. Likewise for data and bss sections.

Some sections are manipulated by 'ld'; others are invented for use of 'as' and have no meaning except during assembly.



File: as.info, Node: Ld Sections, Next: As Sections, Prev: Secs Background, Up: Sections

4.2 Linker Sections

=====

'ld' deals with just four kinds of sections, summarized below.

named sections

text section

data section

These sections hold your program. 'as' and 'ld' treat them as separate but equal sections. Anything you can say of one section is true of another. When the program is running, however, it is customary for the text section to be unalterable. The text section is often shared among processes: it contains instructions, constants and the like. The data section of a running program is usually alterable: for example, C variables would be stored in the data section.

bss section

This section contains zeroed bytes when your program begins running. It is used to hold uninitialized variables or common storage. The length of each partial program's bss section is important, but because it starts out containing zeroed bytes there is no need to store explicit zero bytes in the object file. The bss section was invented to eliminate those explicit zeros from object files.

absolute section

Address 0 of this section is always "relocated" to runtime address 0. This is useful if you want to refer to an address that 'ld' must not change when relocating. In this sense we speak of absolute addresses being "unrelocatable": they do not change during relocation.

undefined section

This "section" is a catch-all for address references to objects not in the preceding sections.

An idealized example of three relocatable sections follows. The example uses the traditional section names '.text' and '.data'. Memory addresses are on the horizontal axis.

```

partial program # 1:  +-----+-----+---+
                    |ttttt|dddd|00|
                    +-----+-----+---+

                    text  data bss
                    seg.  seg. seg.

partial program # 2:  +---+---+---+
                    |TTT|DDD|000|
                    +---+---+---+

linked program:      +--+---+-----+---+-----+---+-----+---+-----+~~
                    | |TTT|ttttt| |dddd|DDD|00000|
                    +--+---+-----+---+-----+---+-----+---+-----+~~

addresses:          0 ...

```



File: as.info, Node: As Sections, Next: Sub-Sections, Prev: Ld Sections, Up: Sections

4.3 Assembler Internal Sections

=====

These sections are meant only for the internal use of 'as'. They have no meaning at run-time. You do not really need to know about these sections for most purposes; but they can be mentioned in 'as' warning messages, so it might be helpful to have an idea of their meanings to 'as'. These sections are used to permit the value of every expression in your assembly language program to be a section-relative address.

ASSEMBLER-INTERNAL-LOGIC-ERROR!

An internal assembler logic error has been found. This means there is a bug in the assembler.

expr section

The assembler stores complex expression internally as combinations of symbols. When it needs to represent an expression as a symbol, it puts it in the expr section.



File: as.info, Node: Sub-Sections, Next: bss, Prev: As Sections, Up: Sections

4.4 Sub-Sections

=====

Assembled bytes conventionally fall into two sections: text and data. You may have separate groups of data in named sections that you want to end up near to each other in the object file, even though they are not contiguous in the assembler source. 'as' allows you to use "subsections" for this purpose. Within each section, there can be numbered subsections with values from 0 to 8192. Objects assembled into the same subsection go into the object file together with other objects in the same subsection. For example, a compiler might want to store

constants in the text section, but might not want to have them interspersed with the program being assembled. In this case, the compiler could issue a `.text 0` before each section of code being output, and a `.text 1` before each group of constants being output.

Subsections are optional. If you do not use subsections, everything goes in subsection number zero.

Each subsection is zero-padded up to a multiple of four bytes. (Subsections may be padded a different amount on different flavors of 'as'.)

Subsections appear in your object file in numeric order, lowest numbered to highest. (All this to be compatible with other people's assemblers.) The object file contains no representation of subsections; 'ld' and other programs that manipulate object files see no trace of them. They just see all your text subsections as a text section, and all your data subsections as a data section.

To specify which subsection you want subsequent statements assembled into, use a numeric argument to specify it, in a `.text EXPRESSION` or a `.data EXPRESSION` statement. When generating COFF output, you can also use an extra subsection argument with arbitrary named sections: `.section NAME, EXPRESSION`. When generating ELF output, you can also use the `.subsection` directive (`*note SubSection::`) to specify a subsection: `.subsection EXPRESSION`. `EXPRESSION` should be an absolute expression (`*note Expressions::`). If you just say `.text` then `.text 0` is assumed. Likewise `.data` means `.data 0`. Assembly begins in `.text 0`. For instance:

```
.text 0      # The default subsection is text 0 anyway.
.ascii "This lives in the first text subsection. *"
.text 1
.ascii "But this lives in the second text subsection."
.data 0
.ascii "This lives in the data section,"
.ascii "in the first data subsection."
.text 0
.ascii "This lives in the first text section,"
.ascii "immediately following the asterisk (*)."
```

Each section has a "location counter" incremented by one for every byte assembled into that section. Because subsections are merely a convenience restricted to 'as' there is no concept of a subsection location counter. There is no way to directly manipulate a location counter--but the `.align` directive changes it, and any label definition captures its current value. The location counter of the section where statements are being assembled is said to be the "active" location counter.



File: as.info, Node: bss, Prev: Sub-Sections, Up: Sections

4.5 bss Section

=====

The bss section is used for local common variable storage. You may allocate address space in the bss section, but you may not dictate data to load into it before your program executes. When your program starts running, all the contents of the bss section are zeroed bytes.

The `'.lcomm'` pseudo-op defines a symbol in the bss section; see `*note '.lcomm': Lcomm`.

The `'.comm'` pseudo-op may be used to declare a common symbol, which is another form of uninitialized symbol; see `*note '.comm': Comm`.

When assembling for a target which supports multiple sections, such as ELF or COFF, you may switch into the `'.bss'` section and define symbols as usual; see `*note '.section': Section`. You may only assemble zero values into the section. Typically the section will only contain symbol definitions and `'.skip'` directives (`*note '.skip': Skip`).



File: as.info, Node: Symbols, Next: Expressions, Prev: Sections, Up: Top

5 Symbols

Symbols are a central concept: the programmer uses symbols to name things, the linker uses symbols to link, and the debugger uses symbols to debug.

`_Warning:` `'as'` does not place symbols in the object file in the same order they were declared. This may break some debuggers.

* Menu:

* Labels::	Labels
* Setting Symbols::	Giving Symbols Other Values
* Symbol Names::	Symbol Names
* Dot::	The Special Dot Symbol
* Symbol Attributes::	Symbol Attributes



File: as.info, Node: Labels, Next: Setting Symbols, Up: Symbols

5.1 Labels

=====

A "label" is written as a symbol immediately followed by a colon `':'`. The symbol then represents the current value of the active location counter, and is, for example, a suitable instruction operand. You are warned if you use the same symbol to represent two different locations: the first definition overrides any other definitions.

On the HPPA, the usual form for a label need not be immediately followed by a colon, but instead must start in column zero. Only one label may be defined on a single line. To work around this, the HPPA version of `'as'` also provides a special directive `'.label'` for defining labels more flexibly.



File: as.info, Node: Setting Symbols, Next: Symbol Names, Prev: Labels, Up: Symbols

5.2 Giving Symbols Other Values

=====

A symbol can be given an arbitrary value by writing a symbol, followed by an equals sign '=', followed by an expression (*note Expressions:). This is equivalent to using the '.set' directive. *Note '.set': Set. In the same way, using a double equals sign '==' here represents an equivalent of the '.eqv' directive. *Note '.eqv': Eqv.

Blackfin does not support symbol assignment with '='.



File: as.info, Node: Symbol Names, Next: Dot, Prev: Setting Symbols, Up: Symbols

5.3 Symbol Names

=====

Symbol names begin with a letter or with one of '._'. On most machines, you can also use '\$' in symbol names; exceptions are noted in *note Machine Dependencies:. That character may be followed by any string of digits, letters, dollar signs (unless otherwise noted for a particular target machine), and underscores.

Case of letters is significant: 'foo' is a different symbol name than 'Foo'.

Symbol names do not start with a digit. An exception to this rule is made for Local Labels. See below.

Multibyte characters are supported. To generate a symbol name containing multibyte characters enclose it within double quotes and use escape codes. cf *Note Strings:. Generating a multibyte symbol name from a label is not currently supported.

Each symbol has exactly one name. Each name in an assembly language program refers to exactly one symbol. You may use that symbol name any number of times in a program.

Local Symbol Names

A local symbol is any symbol beginning with certain local label prefixes. By default, the local label prefix is '.L' for ELF systems or 'L' for traditional a.out systems, but each target may have its own set of local label prefixes. On the HPPA local symbols begin with 'L\$'.

Local symbols are defined and used within the assembler, but they are normally not saved in object files. Thus, they are not visible when debugging. You may use the '-L' option (*note Include Local Symbols: L.) to retain the local symbols in the object files.

Local Labels

Local labels are different from local symbols. Local labels help compilers and programmers use names temporarily. They create symbols which are guaranteed to be unique over the entire scope of the input source code and which can be referred to by a simple notation. To define a local label, write a label of the form 'N:' (where N represents any non-negative integer). To refer to the most recent previous definition of that label write 'Nb', using the same number as when you defined the label. To refer to the next definition of a local label,

write 'Nf'. The 'b' stands for "backwards" and the 'f' stands for "forwards".

There is no restriction on how you can use these labels, and you can reuse them too. So that it is possible to repeatedly define the same local label (using the same number 'N'), although you can only refer to the most recently defined local label of that number (for a backwards reference) or the next definition of a specific local label for a forward reference. It is also worth noting that the first 10 local labels ('0:'...'9:') are implemented in a slightly more efficient manner than the others.

Here is an example:

```
1:      branch 1f
2:      branch 1b
1:      branch 2f
2:      branch 1b
```

Which is the equivalent of:

```
label_1: branch label_3
label_2: branch label_1
label_3: branch label_4
label_4: branch label_3
```

Local label names are only a notational device. They are immediately transformed into more conventional symbol names before the assembler uses them. The symbol names are stored in the symbol table, appear in error messages, and are optionally emitted to the object file. The names are constructed using these parts:

'_local label prefix_'

All local symbols begin with the system-specific local label prefix. Normally both 'as' and 'ld' forget symbols that start with the local label prefix. These labels are used for symbols you are never intended to see. If you use the '-L' option then 'as' retains these symbols in the object file. If you also instruct 'ld' to retain these symbols, you may use them in debugging.

'NUMBER'

This is the number that was used in the local label definition. So if the label is written '55:' then the number is '55'.

'C-B'

This unusual character is included so you do not accidentally invent a symbol of the same name. The character has ASCII value of '\002' (control-B).

'_ordinal number_'

This is a serial number to keep the labels distinct. The first definition of '0:' gets the number '1'. The 15th definition of '0:' gets the number '15', and so on. Likewise the first definition of '1:' gets the number '1' and its 15th definition gets '15' as well.

So for example, the first '1:' may be named '.L1C-B1', and the 44th '3:' may be named '.L3C-B44'.

Dollar Local Labels

On some targets 'as' also supports an even more local form of local labels called dollar labels. These labels go out of scope (i.e., they become undefined) as soon as a non-local label is defined. Thus they remain valid for only a small region of the input source code. Normal local labels, by contrast, remain in scope for the entire file, or until they are redefined by another occurrence of the same local label.

Dollar labels are defined in exactly the same way as ordinary local labels, except that they have a dollar sign suffix to their numeric value, e.g., '55\$:'.

They can also be distinguished from ordinary local labels by their transformed names which use ASCII character '\001' (control-A) as the magic character to distinguish them from ordinary labels. For example, the fifth definition of '6\$' may be named '.L6'C-A'5'.



File: as.info, Node: Dot, Next: Symbol Attributes, Prev: Symbol Names, Up: Symbols

5.4 The Special Dot Symbol

=====

The special symbol '.' refers to the current address that 'as' is assembling into. Thus, the expression 'melvin: .long .' defines 'melvin' to contain its own address. Assigning a value to '.' is treated the same as a '.org' directive. Thus, the expression ' .=.+4' is the same as saying '.space 4'.



File: as.info, Node: Symbol Attributes, Prev: Dot, Up: Symbols

5.5 Symbol Attributes

=====

Every symbol has, as well as its name, the attributes "Value" and "Type". Depending on output format, symbols can also have auxiliary attributes.

If you use a symbol without defining it, 'as' assumes zero for all these attributes, and probably won't warn you. This makes the symbol an externally defined symbol, which is generally what you would want.

* Menu:

* Symbol Value::	Value
* Symbol Type::	Type
* a.out Symbols::	Symbol Attributes: 'a.out'
* COFF Symbols::	Symbol Attributes for COFF
* SOM Symbols::	Symbol Attributes for SOM



File: as.info, Node: Symbol Value, Next: Symbol Type, Up: Symbol Attributes

5.5.1 Value

The value of a symbol is (usually) 32 bits. For a symbol which labels a location in the text, data, bss or absolute sections the value is the number of addresses from the start of that section to the label. Naturally for text, data and bss sections the value of a symbol changes as 'ld' changes section base addresses during linking. Absolute symbols' values do not change during linking: that is why they are called absolute.

The value of an undefined symbol is treated in a special way. If it is 0 then the symbol is not defined in this assembler source file, and 'ld' tries to determine its value from other files linked into the same program. You make this kind of symbol simply by mentioning a symbol name without defining it. A non-zero value represents a '.comm' common declaration. The value is how much common storage to reserve, in bytes (addresses). The symbol refers to the first address of the allocated storage.



File: as.info, Node: Symbol Type, Next: a.out Symbols, Prev: Symbol Value, Up: Symbol Attributes

5.5.2 Type

The type attribute of a symbol contains relocation (section) information, any flag settings indicating that a symbol is external, and (optionally), other information for linkers and debuggers. The exact format depends on the object-code output format in use.



File: as.info, Node: a.out Symbols, Next: COFF Symbols, Prev: Symbol Type, Up: Symbol Attributes

5.5.3 Symbol Attributes: 'a.out'

* Menu:

* Symbol Desc::	Descriptor
* Symbol Other::	Other



File: as.info, Node: Symbol Desc, Next: Symbol Other, Up: a.out Symbols

5.5.3.1 Descriptor

.....

This is an arbitrary 16-bit value. You may establish a symbol's descriptor value by using a '.desc' statement (*note '.desc': Desc.). A descriptor value means nothing to 'as'.



File: as.info, Node: Symbol Other, Prev: Symbol Desc, Up: a.out Symbols

5.5.3.2 Other

.....

This is an arbitrary 8-bit value. It means nothing to 'as'.

US

File: as.info, Node: COFF Symbols, Next: SOM Symbols, Prev: a.out Symbols, Up: Symbol Attributes

5.5.4 Symbol Attributes for COFF

The COFF format supports a multitude of auxiliary symbol attributes; like the primary symbol attributes, they are set between `'.def'` and `'.endef'` directives.

5.5.4.1 Primary Attributes

.....

The symbol name is set with `'.def'`; the value and type, respectively, with `'.val'` and `'.type'`.

5.5.4.2 Auxiliary Attributes

.....

The `'as'` directives `'.dim'`, `'.line'`, `'.scl'`, `'.size'`, `'.tag'`, and `'.weak'` can generate auxiliary symbol table information for COFF.

US

File: as.info, Node: SOM Symbols, Prev: COFF Symbols, Up: Symbol Attributes

5.5.5 Symbol Attributes for SOM

The SOM format for the HPPA supports a multitude of symbol attributes set with the `'.EXPORT'` and `'.IMPORT'` directives.

The attributes are described in 'HP9000 Series 800 Assembly Language Reference Manual' (HP 92432-90001) under the `'IMPORT'` and `'EXPORT'` assembler directive documentation.

US

File: as.info, Node: Expressions, Next: Pseudo Ops, Prev: Symbols, Up: Top

6 Expressions

An "expression" specifies an address or numeric value. Whitespace may precede and/or follow an expression.

The result of an expression must be an absolute number, or else an offset into a particular section. If an expression is not absolute, and there is not enough information when `'as'` sees the expression to know its section, a second pass over the source program might be necessary to interpret the expression--but the second pass is currently not implemented. `'as'` aborts with an error message in this situation.

* Menu:

* Empty Exprs:: Empty Expressions
* Integer Exprs:: Integer Expressions

US

File: as.info, Node: Empty Exprs, Next: Integer Exprs, Up: Expressions

6.1 Empty Expressions

=====

An empty expression has no value: it is just whitespace or null. Wherever an absolute expression is required, you may omit the expression, and 'as' assumes a value of (absolute) 0. This is compatible with other assemblers.



File: as.info, Node: Integer Exprs, Prev: Empty Exprs, Up: Expressions

6.2 Integer Expressions

=====

An "integer expression" is one or more `_arguments_` delimited by `_operators_`.

* Menu:

* Arguments::	Arguments
* Operators::	Operators
* Prefix Ops::	Prefix Operators
* Infix Ops::	Infix Operators



File: as.info, Node: Arguments, Next: Operators, Up: Integer Exprs

6.2.1 Arguments

"Arguments" are symbols, numbers or subexpressions. In other contexts arguments are sometimes called "arithmetic operands". In this manual, to avoid confusing them with the "instruction operands" of the machine language, we use the term "argument" to refer to parts of expressions only, reserving the word "operand" to refer only to machine instruction operands.

Symbols are evaluated to yield {SECTION NNN} where SECTION is one of text, data, bss, absolute, or undefined. NNN is a signed, 2's complement 32 bit integer.

Numbers are usually integers.

A number can be a flonum or bignum. In this case, you are warned that only the low order 32 bits are used, and 'as' pretends these 32 bits are an integer. You may write integer-manipulating instructions that act on exotic constants, compatible with other assemblers.

Subexpressions are a left parenthesis '(' followed by an integer expression, followed by a right parenthesis ')'; or a prefix operator followed by an argument.



File: as.info, Node: Operators, Next: Prefix Ops, Prev: Arguments, Up: Integer Exprs

6.2.2 Operators

"Operators" are arithmetic functions, like '+' or '%'. Prefix operators are followed by an argument. Infix operators appear between their arguments. Operators may be preceded and/or followed by whitespace.



File: as.info, Node: Prefix Ops, Next: Infix Ops, Prev: Operators, Up: Integer Exprs

6.2.3 Prefix Operator

'as' has the following "prefix operators". They each take one argument, which must be absolute.

'-'
"Negation". Two's complement negation.

'~'
"Complementation". Bitwise not.



File: as.info, Node: Infix Ops, Prev: Prefix Ops, Up: Integer Exprs

6.2.4 Infix Operators

"Infix operators" take two arguments, one on either side. Operators have precedence, but operations with equal precedence are performed left to right. Apart from '+' or '-', both arguments must be absolute, and the result is absolute.

1. Highest Precedence

'*'
"Multiplication".

/'
"Division". Truncation is the same as the C operator '/'

'%'
"Remainder".

'<<'
"Shift Left". Same as the C operator '<<'.
'>>'
"Shift Right". Same as the C operator '>>'.

2. Intermediate precedence

'|'
"Bitwise Inclusive Or".

'&'
"Bitwise And".

'^'

"Bitwise Exclusive Or".

'!'

"Bitwise Or Not".

3. Low Precedence

'+'

"Addition". If either argument is absolute, the result has the section of the other argument. You may not add together arguments from different sections.

'-'

"Subtraction". If the right argument is absolute, the result has the section of the left argument. If both arguments are in the same section, the result is absolute. You may not subtract arguments from different sections.

'=='

"Is Equal To"

'<>'

'!='

"Is Not Equal To"

'<'

"Is Less Than"

'>'

"Is Greater Than"

'>='

"Is Greater Than Or Equal To"

'<='

"Is Less Than Or Equal To"

The comparison operators can be used as infix operators. A true results has a value of -1 whereas a false result has a value of 0. Note, these operators perform signed comparisons.

4. Lowest Precedence

'&&'

"Logical And".

'||'

"Logical Or".

These two logical operations can be used to combine the results of sub expressions. Note, unlike the comparison operators a true result returns a value of 1 but a false results does still return 0. Also note that the logical or operator has a slightly lower precedence than logical and.

In short, it's only meaningful to add or subtract the `_offsets_` in an address; you can only have a defined section in one of the two arguments.



File: as.info, Node: Pseudo Ops, Next: Object Attributes, Prev: Expressions, Up: Top

All assembler directives have names that begin with a period ('.'). The names are case insensitive for most targets, and usually written in lower case.

This chapter discusses directives that are available regardless of the target machine configuration for the GNU assembler. Some machine configurations provide additional directives. *Note Machine Dependencies:.

* Menu:

```
* Abort::                '.abort'
* ABORT (COFF)::        '.ABORT'

* Align::               '.align ABS-EXPR , ABS-EXPR'
* Altmacro::           '.altmacro'
* Ascii::              '.ascii "STRING"...'
* Asciz::              '.asciz "STRING"...'
* Balign::             '.balign ABS-EXPR , ABS-EXPR'
* Bundle directives::  '.bundle_align_mode ABS-EXPR', etc
* Byte::               '.byte EXPRESSIONS'
* CFI directives::     '.cfi_startproc [simple]', '.cfi_endproc', etc.
* Comm::               '.comm SYMBOL , LENGTH '
* Data::               '.data SUBSECTION'
* Def::                '.def NAME'
* Desc::               '.desc SYMBOL, ABS-EXPRESSION'
* Dim::                '.dim'

* Double::             '.double FLONUMS'
* Eject::              '.eject'
* Else::               '.else'
* Elseif::             '.elseif'
* End::                '.end'
* Endef::              '.endef'

* Endfunc::            '.endfunc'
* Endif::              '.endif'
* Equ::                '.equ SYMBOL, EXPRESSION'
* Equiv::              '.equiv SYMBOL, EXPRESSION'
* Eqv::                '.eqv SYMBOL, EXPRESSION'
* Err::                '.err'
* Error::              '.error STRING'
* Exitm::              '.exitm'
* Extern::             '.extern'
* Fail::               '.fail'
* File::               '.file'
* Fill::               '.fill REPEAT , SIZE , VALUE'
* Float::              '.float FLONUMS'
* Func::               '.func'
* Global::             '.global SYMBOL', '.globl SYMBOL'
* Gnu_attribute::     '.gnu_attribute TAG,VALUE'
* Hidden::             '.hidden NAMES'

* hword::              '.hword EXPRESSIONS'
* Ident::              '.ident'
* If::                 '.if ABSOLUTE EXPRESSION'
* Incbin::             '.incbin "FILE"[,SKIP[,COUNT]]'
```

```

* Include::          '.include "FILE"'
* Int::             '.int EXPRESSIONS'
* Internal::        '.internal NAMES'

* Irp::            '.irp SYMBOL,VALUES'...
* Irpc::           '.irpc SYMBOL,VALUES'...
* Lcomm::          '.lcomm SYMBOL , LENGTH'
* Lflags::         '.lflags'
* Line::           '.line LINE-NUMBER'

* Linkonce::       '.linkonce [TYPE]'
* List::           '.list'
* Ln::             '.ln LINE-NUMBER'
* Loc::            '.loc FILENO LINENO'
* Loc_mark_labels:: '.loc_mark_labels ENABLE'
* Local::          '.local NAMES'

* Long::           '.long EXPRESSIONS'

* Macro::          '.macro NAME ARGS'...
* MRI::            '.mri VAL'
* Noaltmacro::     '.noaltmacro'
* Nolist::         '.nolist'
* Octa::           '.octa BIGNUMS'
* Offset::         '.offset LOC'
* Org::            '.org NEW-LC, FILL'
* P2align::        '.p2align ABS-EXPR, ABS-EXPR, ABS-EXPR'
* PopSection::     '.popsection'
* Previous::       '.previous'

* Print::          '.print STRING'
* Protected::      '.protected NAMES'

* Psize::          '.psize LINES, COLUMNS'
* Purgem::         '.purgem NAME'
* PushSection::    '.pushsection NAME'

* Quad::           '.quad BIGNUMS'
* Reloc::          '.reloc OFFSET, RELOC_NAME[, EXPRESSION]'
* Rept::           '.rept COUNT'
* Sbtatl::         '.sbtatl "SUBHEADING"'
* Scl::            '.scl CLASS'
* Section::        '.section NAME[, FLAGS]'

* Set::            '.set SYMBOL, EXPRESSION'
* Short::          '.short EXPRESSIONS'
* Single::         '.single FLONUMS'
* Size::           '.size [NAME , EXPRESSION]'
* Skip::           '.skip SIZE , FILL'

* Sleb128::        '.sleb128 EXPRESSIONS'
* Space::          '.space SIZE , FILL'
* Stab::           '.stabd, .stabn, .stabs'

* String::         '.string "STR"', '.string8 "STR"', '.string16 "STR"',
'.string32 "STR"', '.string64 "STR"'
* Struct::         '.struct EXPRESSION'
* SubSection::     '.subsection'
* Symver::         '.symver NAME,NAME2@NODENAME'

```

```

* Tag::                '.tag STRUCTNAME'
* Text::               '.text SUBSECTION'
* Title::              '.title "HEADING"'
* Type::               '.type <INT | NAME , TYPE DESCRIPTION>'
* Uleb128::            '.uleb128 EXPRESSIONS'
* Val::                '.val ADDR'
* Version::            '.version "STRING"'
* VTableEntry::        '.vtable_entry TABLE, OFFSET'
* VTableInherit::      '.vtable_inherit CHILD, PARENT'
* Warning::            '.warning STRING'
* Weak::               '.weak NAMES'
* Weakref::            '.weakref ALIAS, SYMBOL'
* Word::               '.word EXPRESSIONS'
* Zero::               '.zero SIZE'
* Deprecated::         Deprecated Directives

```



File: as.info, Node: Abort, Next: ABORT (COFF), Up: Pseudo Ops

7.1 '.abort'

=====

This directive stops the assembly immediately. It is for compatibility with other assemblers. The original idea was that the assembly language source would be piped into the assembler. If the sender of the source quit, it could use this directive tells 'as' to quit also. One day '.abort' will not be supported.



File: as.info, Node: ABORT (COFF), Next: Align, Prev: Abort, Up: Pseudo Ops

7.2 '.ABORT' (COFF)

=====

When producing COFF output, 'as' accepts this directive as a synonym for '.abort'.



File: as.info, Node: Align, Next: Altmacro, Prev: ABORT (COFF), Up: Pseudo Ops

7.3 '.align ABS-EXPR, ABS-EXPR, ABS-EXPR'

=====

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the alignment required, as described below.

The second expression (also absolute) gives the fill value to be stored in the padding bytes. It (and the comma) may be omitted. If it is omitted, the padding bytes are normally zero. However, on some systems, if the section is marked as containing code and the fill value is omitted, the space is filled with no-op instructions.

The third expression is also absolute, and is also optional. If it

is present, it is the maximum number of bytes that should be skipped by this alignment directive. If doing the alignment would require skipping more bytes than the specified maximum, then the alignment is not done at all. You can omit the fill value (the second argument) entirely by simply using two commas after the required alignment; this can be useful if you want the alignment to be filled with no-op instructions when appropriate.

The way the required alignment is specified varies from system to system. For the arc, hppa, i386 using ELF, i860, iq2000, m68k, or1k, s390, sparc, tic4x, tic80 and xtensa, the first expression is the alignment request in bytes. For example `'.align 8'` advances the location counter until it is a multiple of 8. If the location counter is already a multiple of 8, no change is needed. For the tic54x, the first expression is the alignment request in words.

For other systems, including ppc, i386 using a.out format, arm and strongarm, it is the number of low-order zero bits the location counter must have after advancement. For example `'.align 3'` advances the location counter until it a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

This inconsistency is due to the different behaviors of the various native assemblers for these systems which GAS must emulate. GAS also provides `'.balign'` and `'.p2align'` directives, described later, which have a consistent behavior across all architectures (but are specific to GAS).



File: as.info, Node: Altmacro, Next: Ascii, Prev: Align, Up: Pseudo Ops

7.4 `'.altmacro'`

=====

Enable alternate macro mode, enabling:

`'LOCAL NAME [, ...]'`

One additional directive, `'LOCAL'`, is available. It is used to generate a string replacement for each of the NAME arguments, and replace any instances of NAME in each macro expansion. The replacement string is unique in the assembly, and different for each separate macro expansion. `'LOCAL'` allows you to write macros that define symbols, without fear of conflict between separate macro expansions.

`'String delimiters'`

You can write strings delimited in these other ways besides

`"STRING"`:

`'STRING'`

You can delimit strings with single-quote characters.

`<STRING>`

You can delimit strings with matching angle brackets.

`'single-character string escape'`

To include any single character literally in a string (even if the character would otherwise have some special meaning), you can prefix the character with `'!'` (an exclamation mark). For example,

you can write '<4.3 !> 5.4!!>' to get the literal text '4.3 > 5.4!'.

'Expression results as strings'

You can write '%EXPR' to evaluate the expression EXPR and use the result as a string.



File: as.info, Node: Ascii, Next: Asciz, Prev: Altmacro, Up: Pseudo Ops

7.5 '.ascii "STRING"'

'ascii' expects zero or more string literals (*note Strings::) separated by commas. It assembles each string (with no automatic trailing zero byte) into consecutive addresses.



File: as.info, Node: Asciz, Next: Balign, Prev: Ascii, Up: Pseudo Ops

7.6 '.asciz "STRING"'

'asciz' is just like 'ascii', but each string is followed by a zero byte. The "z" in 'asciz' stands for "zero".



File: as.info, Node: Balign, Next: Bundle directives, Prev: Asciz, Up: Pseudo Ops

7.7 '.balign[w] ABS-EXPR, ABS-EXPR, ABS-EXPR'

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the alignment request in bytes. For example '.balign 8' advances the location counter until it is a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

The second expression (also absolute) gives the fill value to be stored in the padding bytes. It (and the comma) may be omitted. If it is omitted, the padding bytes are normally zero. However, on some systems, if the section is marked as containing code and the fill value is omitted, the space is filled with no-op instructions.

The third expression is also absolute, and is also optional. If it is present, it is the maximum number of bytes that should be skipped by this alignment directive. If doing the alignment would require skipping more bytes than the specified maximum, then the alignment is not done at all. You can omit the fill value (the second argument) entirely by simply using two commas after the required alignment; this can be useful if you want the alignment to be filled with no-op instructions when appropriate.

The '.balignw' and '.balignl' directives are variants of the '.balign' directive. The '.balignw' directive treats the fill pattern as a two byte word value. The '.balignl' directives treats the fill pattern as a four byte longword value. For example, '.balignw 4,0x368d' will align to a multiple of 4. If it skips two bytes, they will be filled in with the value 0x368d (the exact placement of the bytes

depends upon the endianness of the processor). If it skips 1 or 3 bytes, the fill value is undefined.



File: as.info, Node: Bundle directives, Next: Byte, Prev: Balign, Up: Pseudo Ops

7.8 Bundle directives

=====

7.8.1 '.bundle_align_mode ABS-EXPR'

'.bundle_align_mode' enables or disables "aligned instruction bundle" mode. In this mode, sequences of adjacent instructions are grouped into fixed-sized "bundles". If the argument is zero, this mode is disabled (which is the default state). If the argument is not zero, it gives the size of an instruction bundle as a power of two (as for the '.p2align' directive, *note P2align:).).

For some targets, it's an ABI requirement that no instruction may span a certain aligned boundary. A "bundle" is simply a sequence of instructions that starts on an aligned boundary. For example, if ABS-EXPR is '5' then the bundle size is 32, so each aligned chunk of 32 bytes is a bundle. When aligned instruction bundle mode is in effect, no single instruction may span a boundary between bundles. If an instruction would start too close to the end of a bundle for the length of that particular instruction to fit within the bundle, then the space at the end of that bundle is filled with no-op instructions so the instruction starts in the next bundle. As a corollary, it's an error if any single instruction's encoding is longer than the bundle size.

7.8.2 '.bundle_lock' and '.bundle_unlock'

The '.bundle_lock' and directive '.bundle_unlock' directives allow explicit control over instruction bundle padding. These directives are only valid when '.bundle_align_mode' has been used to enable aligned instruction bundle mode. It's an error if they appear when '.bundle_align_mode' has not been used at all, or when the last directive was '.bundle_align_mode 0'.

For some targets, it's an ABI requirement that certain instructions may appear only as part of specified permissible sequences of multiple instructions, all within the same bundle. A pair of '.bundle_lock' and '.bundle_unlock' directives define a "bundle-locked" instruction sequence. For purposes of aligned instruction bundle mode, a sequence starting with '.bundle_lock' and ending with '.bundle_unlock' is treated as a single instruction. That is, the entire sequence must fit into a single bundle and may not span a bundle boundary. If necessary, no-op instructions will be inserted before the first instruction of the sequence so that the whole sequence starts on an aligned bundle boundary. It's an error if the sequence is longer than the bundle size.

For convenience when using '.bundle_lock' and '.bundle_unlock' inside assembler macros (*note Macro:), bundle-locked sequences may be nested. That is, a second '.bundle_lock' directive before the next '.bundle_unlock' directive has no effect except that it must be matched by another closing '.bundle_unlock' so that there is the same number of '.bundle_lock' and '.bundle_unlock' directives.

US

File: as.info, Node: Byte, Next: CFI directives, Prev: Bundle directives, Up: Pseudo Ops

7.9 '.byte EXPRESSIONS'

=====

'.byte' expects zero or more expressions, separated by commas. Each expression is assembled into the next byte.

US

File: as.info, Node: CFI directives, Next: Comm, Prev: Byte, Up: Pseudo Ops

7.10 CFI directives

=====

7.10.1 '.cfi_sections SECTION_LIST'

'.cfi_sections' may be used to specify whether CFI directives should emit '.eh_frame' section and/or '.debug_frame' section. If SECTION_LIST is '.eh_frame', '.eh_frame' is emitted, if SECTION_LIST is '.debug_frame', '.debug_frame' is emitted. To emit both use '.eh_frame, .debug_frame'. The default if this directive is not used is '.cfi_sections .eh_frame'.

On targets that support compact unwinding tables these can be generated by specifying '.eh_frame_entry' instead of '.eh_frame'.

Some targets may support an additional name, such as '.c6xabi.exidx' which is used by the target.

The '.cfi_sections' directive can be repeated, with the same or different arguments, provided that CFI generation has not yet started. Once CFI generation has started however the section list is fixed and any attempts to redefine it will result in an error.

7.10.2 '.cfi_startproc [simple]'

'.cfi_startproc' is used at the beginning of each function that should have an entry in '.eh_frame'. It initializes some internal data structures. Don't forget to close the function by '.cfi_endproc'.

Unless '.cfi_startproc' is used along with parameter 'simple' it also emits some architecture dependent initial CFI instructions.

7.10.3 '.cfi_endproc'

'.cfi_endproc' is used at the end of a function where it closes its unwind entry previously opened by '.cfi_startproc', and emits it to '.eh_frame'.

7.10.4 '.cfi_personality ENCODING [, EXP]'

'.cfi_personality' defines personality routine and its encoding.

ENCODING must be a constant determining how the personality should be encoded. If it is 255 ('DW_EH_PE_omit'), second argument is not present, otherwise second argument should be a constant or a symbol name. When using indirect encodings, the symbol provided should be the location where personality can be loaded from, not the personality routine itself. The default after '.cfi_startproc' is '.cfi_personality 0xff', no personality routine.

7.10.5 '.cfi_personality_id ID'

'cfi_personality_id' defines a personality routine by its index as defined in a compact unwinding format. Only valid when generating compact EH frames (i.e. with '.cfi_sections eh_frame_entry').

7.10.6 '.cfi_fde_data [OPCODE1 [, ...]]'

'cfi_fde_data' is used to describe the compact unwind opcodes to be used for the current function. These are emitted inline in the '.eh_frame_entry' section if small enough and there is no LSDA, or in the '.gnu.extab' section otherwise. Only valid when generating compact EH frames (i.e. with '.cfi_sections eh_frame_entry').

7.10.7 '.cfi_lsdas ENCODING [, EXP]'

'cfi_lsdas' defines LSDA and its encoding. ENCODING must be a constant determining how the LSDA should be encoded. If it is 255 ('DW_EH_PE_omit'), the second argument is not present, otherwise the second argument should be a constant or a symbol name. The default after '.cfi_startproc' is '.cfi_lsdas 0xff', meaning that no LSDA is present.

7.10.8 '.cfi_inline_lsdas' [ALIGN]

'cfi_inline_lsdas' marks the start of a LSDA data section and switches to the corresponding '.gnu.extab' section. Must be preceded by a CFI block containing a '.cfi_lsdas' directive. Only valid when generating compact EH frames (i.e. with '.cfi_sections eh_frame_entry').

The table header and unwinding opcodes will be generated at this point, so that they are immediately followed by the LSDA data. The symbol referenced by the '.cfi_lsdas' directive should still be defined in case a fallback FDE based encoding is used. The LSDA data is terminated by a section directive.

The optional ALIGN argument specifies the alignment required. The alignment is specified as a power of two, as with the '.p2align' directive.

7.10.9 '.cfi_def_cfa REGISTER, OFFSET'

'cfi_def_cfa' defines a rule for computing CFA as: take address from REGISTER and add OFFSET to it.

7.10.10 '.cfi_def_cfa_register REGISTER'

'.cfi_def_cfa_register' modifies a rule for computing CFA. From now on REGISTER will be used instead of the old one. Offset remains the same.

7.10.11 '.cfi_def_cfa_offset OFFSET'

'.cfi_def_cfa_offset' modifies a rule for computing CFA. Register remains the same, but OFFSET is new. Note that it is the absolute offset that will be added to a defined register to compute CFA address.

7.10.12 '.cfi_adjust_cfa_offset OFFSET'

Same as '.cfi_def_cfa_offset' but OFFSET is a relative value that is added/subtracted from the previous offset.

7.10.13 '.cfi_offset REGISTER, OFFSET'

Previous value of REGISTER is saved at offset OFFSET from CFA.

7.10.14 '.cfi_val_offset REGISTER, OFFSET'

Previous value of REGISTER is CFA + OFFSET.

7.10.15 '.cfi_rel_offset REGISTER, OFFSET'

Previous value of REGISTER is saved at offset OFFSET from the current CFA register. This is transformed to '.cfi_offset' using the known displacement of the CFA register from the CFA. This is often easier to use, because the number will match the code it's annotating.

7.10.16 '.cfi_register REGISTER1, REGISTER2'

Previous value of REGISTER1 is saved in register REGISTER2.

7.10.17 '.cfi_restore REGISTER'

'.cfi_restore' says that the rule for REGISTER is now the same as it was at the beginning of the function, after all initial instruction added by '.cfi_startproc' were executed.

7.10.18 '.cfi_undefined REGISTER'

From now on the previous value of REGISTER can't be restored anymore.

7.10.19 '.cfi_same_value REGISTER'

Current value of REGISTER is the same like in the previous frame, i.e. no restoration needed.

7.10.20 '.cfi_remember_state' and '.cfi_restore_state'

'.cfi_remember_state' pushes the set of rules for every register onto an implicit stack, while '.cfi_restore_state' pops them off the stack and places them in the current row. This is useful for situations where you have multiple '.cfi_*' directives that need to be undone due to the control flow of the program. For example, we could have something like this (assuming the CFA is the value of 'rbp'):

```

        je label
        popq %rbx
        .cfi_restore %rbx
        popq %r12
        .cfi_restore %r12
        popq %rbp
        .cfi_restore %rbp
        .cfi_def_cfa %rsp, 8
        ret
label:
        /* Do something else */

```

Here, we want the '.cfi' directives to affect only the rows corresponding to the instructions before 'label'. This means we'd have to add multiple '.cfi' directives after 'label' to recreate the original save locations of the registers, as well as setting the CFA back to the value of 'rbp'. This would be clumsy, and result in a larger binary size. Instead, we can write:

```

        je label
        popq %rbx
        .cfi_remember_state
        .cfi_restore %rbx
        popq %r12
        .cfi_restore %r12
        popq %rbp
        .cfi_restore %rbp
        .cfi_def_cfa %rsp, 8
        ret
label:
        .cfi_restore_state
        /* Do something else */

```

That way, the rules for the instructions after 'label' will be the same as before the first '.cfi_restore' without having to use multiple '.cfi' directives.

7.10.21 '.cfi_return_column REGISTER'

Change return column REGISTER, i.e. the return address is either directly in REGISTER or can be accessed by rules for REGISTER.

7.10.22 '.cfi_signal_frame'

Mark current function as signal trampoline.

7.10.23 '.cfi_window_save'

 SPARC register window has been saved.

7.10.24 `'.cfi_escape' EXPRESSION[, ...]`

Allows the user to add arbitrary bytes to the unwind info. One might use this to add OS-specific CFI opcodes, or generic CFI opcodes that GAS does not yet support.

7.10.25 `'.cfi_val_encoded_addr REGISTER, ENCODING, LABEL'`

The current value of REGISTER is LABEL. The value of LABEL will be encoded in the output file according to ENCODING; see the description of `'.cfi_personality'` for details on this encoding.

The usefulness of equating a register to a fixed label is probably limited to the return address register. Here, it can be useful to mark a code segment that has only one return address which is reached by a direct branch and no copy of the return address exists in memory or another register.



File: as.info, Node: Comm, Next: Data, Prev: CFI directives, Up: Pseudo Ops

7.11 `'.comm SYMBOL , LENGTH'`

=====

`'.comm'` declares a common symbol named SYMBOL. When linking, a common symbol in one object file may be merged with a defined or common symbol of the same name in another object file. If `'ld'` does not see a definition for the symbol—just one or more common symbols—then it will allocate LENGTH bytes of uninitialized memory. LENGTH must be an absolute expression. If `'ld'` sees multiple common symbols with the same name, and they do not all have the same size, it will allocate space using the largest size.

When using ELF or (as a GNU extension) PE, the `'.comm'` directive takes an optional third argument. This is the desired alignment of the symbol, specified for ELF as a byte boundary (for example, an alignment of 16 means that the least significant 4 bits of the address should be zero), and for PE as a power of two (for example, an alignment of 5 means aligned to a 32-byte boundary). The alignment must be an absolute expression, and it must be a power of two. If `'ld'` allocates uninitialized memory for the common symbol, it will use the alignment when placing the symbol. If no alignment is specified, `'as'` will set the alignment to the largest power of two less than or equal to the size of the symbol, up to a maximum of 16 on ELF, or the default section alignment of 4 on PE(1).

The syntax for `'.comm'` differs slightly on the HPPA. The syntax is `'SYMBOL .comm, LENGTH'`; SYMBOL is optional.

----- Footnotes -----

(1) This is not the same as the executable image file alignment controlled by `'ld'`'s `'--section-alignment'` option; image file sections

in PE are aligned to multiples of 4096, which is far too large an alignment for ordinary variables. It is rather the default alignment for (non-debug) sections within object ('*.o') files, which are less strictly aligned.

US

File: as.info, Node: Data, Next: Def, Prev: Comm, Up: Pseudo Ops

7.12 '.data SUBSECTION'

=====

'.data' tells 'as' to assemble the following statements onto the end of the data subsection numbered SUBSECTION (which is an absolute expression). If SUBSECTION is omitted, it defaults to zero.

US

File: as.info, Node: Def, Next: Desc, Prev: Data, Up: Pseudo Ops

7.13 '.def NAME'

=====

Begin defining debugging information for a symbol NAME; the definition extends until the '.endef' directive is encountered.

US

File: as.info, Node: Desc, Next: Dim, Prev: Def, Up: Pseudo Ops

7.14 '.desc SYMBOL, ABS-EXPRESSION'

=====

This directive sets the descriptor of the symbol (*note Symbol Attributes::) to the low 16 bits of an absolute expression.

The '.desc' directive is not available when 'as' is configured for COFF output; it is only for 'a.out' or 'b.out' object format. For the sake of compatibility, 'as' accepts it, but produces no output, when configured for COFF.

US

File: as.info, Node: Dim, Next: Double, Prev: Desc, Up: Pseudo Ops

7.15 '.dim'

=====

This directive is generated by compilers to include auxiliary debugging information in the symbol table. It is only permitted inside '.def'/'endef' pairs.

US

File: as.info, Node: Double, Next: Eject, Prev: Dim, Up: Pseudo Ops

7.16 '.double FLONUMS'

=====

'.double' expects zero or more flonums, separated by commas. It assembles floating point numbers. The exact kind of floating point numbers emitted depends on how 'as' is configured. *Note Machine Dependencies::.

US

File: as.info, Node: Eject, Next: Else, Prev: Double, Up: Pseudo Ops

7.17 '.eject'

=====

Force a page break at this point, when generating assembly listings.

US

File: as.info, Node: Else, Next: Elseif, Prev: Eject, Up: Pseudo Ops

7.18 '.else'

=====

'.else' is part of the 'as' support for conditional assembly; see *note '.if': If. It marks the beginning of a section of code to be assembled if the condition for the preceding '.if' was false.

US

File: as.info, Node: Elseif, Next: End, Prev: Else, Up: Pseudo Ops

7.19 '.elseif'

=====

'.elseif' is part of the 'as' support for conditional assembly; see *note '.if': If. It is shorthand for beginning a new '.if' block that would otherwise fill the entire '.else' section.

US

File: as.info, Node: End, Next: Endef, Prev: Elseif, Up: Pseudo Ops

7.20 '.end'

=====

'.end' marks the end of the assembly file. 'as' does not process anything in the file past the '.end' directive.

US

File: as.info, Node: Endef, Next: Endfunc, Prev: End, Up: Pseudo Ops

7.21 '.endef'

=====

This directive flags the end of a symbol definition begun with '.def'.

US

File: as.info, Node: Endfunc, Next: Endif, Prev: Endef, Up: Pseudo Ops

7.22 '.endfunc'

=====

'.endfunc' marks the end of a function specified with '.func'.

US

File: as.info, Node: Endif, Next: Equ, Prev: Endfunc, Up: Pseudo Ops

7.23 '.endif'

=====

'`.endif`' is part of the '`as`' support for conditional assembly; it marks the end of a block of code that is only assembled conditionally. *Note '`.if`': If.

US

File: `as.info`, Node: `Equ`, Next: `Equiv`, Prev: `Endif`, Up: `Pseudo Ops`

7.24 '`.equ SYMBOL, EXPRESSION`'

=====

This directive sets the value of `SYMBOL` to `EXPRESSION`. It is synonymous with '`.set`'; see *note '`.set`': Set.

The syntax for '`equ`' on the HPPA is '`SYMBOL .equ EXPRESSION`'.

The syntax for '`equ`' on the Z80 is '`SYMBOL equ EXPRESSION`'. On the Z80 it is an error if `SYMBOL` is already defined, but the symbol is not protected from later redefinition. Compare *note `Equiv::`.

US

File: `as.info`, Node: `Equiv`, Next: `Eqv`, Prev: `Equ`, Up: `Pseudo Ops`

7.25 '`.equiv SYMBOL, EXPRESSION`'

=====

The '`.equiv`' directive is like '`.equ`' and '`.set`', except that the assembler will signal an error if `SYMBOL` is already defined. Note a symbol which has been referenced but not actually defined is considered to be undefined.

Except for the contents of the error message, this is roughly equivalent to

```
.ifdef SYM
.err
.endif
.equ SYM,VAL
```

plus it protects the symbol from later redefinition.

US

File: `as.info`, Node: `Eqv`, Next: `Err`, Prev: `Equiv`, Up: `Pseudo Ops`

7.26 '`.eqv SYMBOL, EXPRESSION`'

=====

The '`.eqv`' directive is like '`.equiv`', but no attempt is made to evaluate the expression or any part of it immediately. Instead each time the resulting symbol is used in an expression, a snapshot of its current value is taken.

US

File: `as.info`, Node: `Err`, Next: `Error`, Prev: `Eqv`, Up: `Pseudo Ops`

7.27 '`.err`'

=====

If '`as`' assembles a '`.err`' directive, it will print an error message and, unless the '`-Z`' option was used, it will not generate an object file. This can be used to signal an error in conditionally compiled code.

US

File: as.info, Node: Error, Next: Exitm, Prev: Err, Up: Pseudo Ops

7.28 '.error "STRING"'

=====

Similarly to '.err', this directive emits an error, but you can specify a string that will be emitted as the error message. If you don't specify the message, it defaults to '".error directive invoked in source file"'. *Note Error and Warning Messages: Errors.

```
.error "This code has not been assembled and tested."
```

US

File: as.info, Node: Exitm, Next: Extern, Prev: Error, Up: Pseudo Ops

7.29 '.exitm'

=====

Exit early from the current macro definition. *Note Macro:.

US

File: as.info, Node: Extern, Next: Fail, Prev: Exitm, Up: Pseudo Ops

7.30 '.extern'

=====

'.extern' is accepted in the source program--for compatibility with other assemblers--but it is ignored. 'as' treats all undefined symbols as external.

US

File: as.info, Node: Fail, Next: File, Prev: Extern, Up: Pseudo Ops

7.31 '.fail EXPRESSION'

=====

Generates an error or a warning. If the value of the EXPRESSION is 500 or more, 'as' will print a warning message. If the value is less than 500, 'as' will print an error message. The message will include the value of EXPRESSION. This can occasionally be useful inside complex nested macros or conditional assembly.

US

File: as.info, Node: File, Next: Fill, Prev: Fail, Up: Pseudo Ops

7.32 '.file'

=====

There are two different versions of the '.file' directive. Targets that support DWARF2 line number information use the DWARF2 version of '.file'. Other targets use the default version.

Default Version

This version of the '.file' directive tells 'as' that we are about to start a new logical file. The syntax is:

.file STRING

STRING is the new file name. In general, the filename is recognized whether or not it is surrounded by quotes ''; but if you wish to specify an empty file name, you must give the quotes-'''. This statement may go away in future: it is only recognized to be compatible with old 'as' programs.

DWARF2 Version

When emitting DWARF2 line number information, '.file' assigns filenames to the '.debug_line' file name table. The syntax is:

.file FILENO FILENAME

The FILENO operand should be a unique positive integer to use as the index of the entry in the table. The FILENAME operand is a C string literal.

The detail of filename indices is exposed to the user because the filename table is shared with the '.debug_info' section of the DWARF2 debugging information, and thus the user must know the exact indices that table entries will have.



File: as.info, Node: Fill, Next: Float, Prev: File, Up: Pseudo Ops

7.33 '.fill REPEAT , SIZE , VALUE'

=====

REPEAT, SIZE and VALUE are absolute expressions. This emits REPEAT copies of SIZE bytes. REPEAT may be zero or more. SIZE may be zero or more, but if it is more than 8, then it is deemed to have the value 8, compatible with other people's assemblers. The contents of each REPEAT bytes is taken from an 8-byte number. The highest order 4 bytes are zero. The lowest order 4 bytes are VALUE rendered in the byte-order of an integer on the computer 'as' is assembling for. Each SIZE bytes in a repetition is taken from the lowest order SIZE bytes of this number. Again, this bizarre behavior is compatible with other people's assemblers.

SIZE and VALUE are optional. If the second comma and VALUE are absent, VALUE is assumed zero. If the first comma and following tokens are absent, SIZE is assumed to be 1.



File: as.info, Node: Float, Next: Func, Prev: Fill, Up: Pseudo Ops

7.34 '.float FLONUMS'

=====

This directive assembles zero or more flonums, separated by commas. It has the same effect as '.single'. The exact kind of floating point numbers emitted depends on how 'as' is configured. *Note Machine Dependencies:..



File: as.info, Node: Func, Next: Global, Prev: Float, Up: Pseudo Ops

7.35 '.func NAME[,LABEL]'

=====

'.func' emits debugging information to denote function NAME, and is ignored unless the file is assembled with debugging enabled. Only '--gstabs[+]' is currently supported. LABEL is the entry point of the function and if omitted NAME prepended with the 'leading char' is used. 'leading char' is usually '_' or nothing, depending on the target. All functions are currently defined to have 'void' return type. The function must be terminated with '.endfunc'.



File: as.info, Node: Global, Next: Gnu_attribute, Prev: Func, Up: Pseudo Ops

7.36 '.global SYMBOL', '.globl SYMBOL'

=====

'.global' makes the symbol visible to 'ld'. If you define SYMBOL in your partial program, its value is made available to other partial programs that are linked with it. Otherwise, SYMBOL takes its attributes from a symbol of the same name from another file linked into the same program.

Both spellings ('.globl' and '.global') are accepted, for compatibility with other assemblers.

On the HPPA, '.global' is not always enough to make it accessible to other partial programs. You may need the HPPA-only '.EXPORT' directive as well. *Note HPPA Assembler Directives: HPPA Directives.



File: as.info, Node: Gnu_attribute, Next: Hidden, Prev: Global, Up: Pseudo Ops

7.37 '.gnu_attribute TAG,VALUE'

=====

Record a GNU object attribute for this file. *Note Object Attributes::.



File: as.info, Node: Hidden, Next: hword, Prev: Gnu_attribute, Up: Pseudo Ops

7.38 '.hidden NAMES'

=====

This is one of the ELF visibility directives. The other two are '.internal' (*note '.internal': Internal.) and '.protected' (*note '.protected': Protected.).

This directive overrides the named symbols default visibility (which is set by their binding: local, global or weak). The directive sets the visibility to 'hidden' which means that the symbols are not visible to other components. Such symbols are always considered to be 'protected' as well.



File: as.info, Node: hword, Next: Ident, Prev: Hidden, Up: Pseudo Ops

7.39 '.hword EXPRESSIONS'

=====

This expects zero or more EXPRESSIONS, and emits a 16 bit number for each.

This directive is a synonym for '.short'; depending on the target architecture, it may also be a synonym for '.word'.

US

File: as.info, Node: Ident, Next: If, Prev: hword, Up: Pseudo Ops

7.40 '.ident'

=====

This directive is used by some assemblers to place tags in object files. The behavior of this directive varies depending on the target. When using the a.out object file format, 'as' simply accepts the directive for source-file compatibility with existing assemblers, but does not emit anything for it. When using COFF, comments are emitted to the '.comment' or '.rdata' section, depending on the target. When using ELF, comments are emitted to the '.comment' section.

US

File: as.info, Node: If, Next: Incbin, Prev: Ident, Up: Pseudo Ops

7.41 '.if ABSOLUTE EXPRESSION'

=====

'.if' marks the beginning of a section of code which is only considered part of the source program being assembled if the argument (which must be an ABSOLUTE EXPRESSION) is non-zero. The end of the conditional section of code must be marked by '.endif' (*note '.endif': Endif.); optionally, you may include code for the alternative condition, flagged by '.else' (*note '.else': Else.). If you have several conditions to check, '.elseif' may be used to avoid nesting blocks if/else within each subsequent '.else' block.

The following variants of '.if' are also supported:

'.ifdef SYMBOL'

Assembles the following section of code if the specified SYMBOL has been defined. Note a symbol which has been referenced but not yet defined is considered to be undefined.

'.ifb TEXT'

Assembles the following section of code if the operand is blank (empty).

'.ifc STRING1,STRING2'

Assembles the following section of code if the two strings are the same. The strings may be optionally quoted with single quotes. If they are not quoted, the first string stops at the first comma, and the second string stops at the end of the line. Strings which contain whitespace should be quoted. The string comparison is case sensitive.

'.ifeq ABSOLUTE EXPRESSION'

Assembles the following section of code if the argument is zero.

```

'.ifeqs STRING1,STRING2'
  Another form of '.ifc'. The strings must be quoted using double
  quotes.

'.ifge ABSOLUTE EXPRESSION'
  Assembles the following section of code if the argument is greater
  than or equal to zero.

'.ifgt ABSOLUTE EXPRESSION'
  Assembles the following section of code if the argument is greater
  than zero.

'.ifle ABSOLUTE EXPRESSION'
  Assembles the following section of code if the argument is less
  than or equal to zero.

'.iflt ABSOLUTE EXPRESSION'
  Assembles the following section of code if the argument is less
  than zero.

'.ifnb TEXT'
  Like '.ifb', but the sense of the test is reversed: this assembles
  the following section of code if the operand is non-blank
  (non-empty).

'.ifnc STRING1,STRING2.'
  Like '.ifc', but the sense of the test is reversed: this assembles
  the following section of code if the two strings are not the same.

'.ifndef SYMBOL'
'.ifnotdef SYMBOL'
  Assembles the following section of code if the specified SYMBOL has
  not been defined. Both spelling variants are equivalent. Note a
  symbol which has been referenced but not yet defined is considered
  to be undefined.

'.ifne ABSOLUTE EXPRESSION'
  Assembles the following section of code if the argument is not
  equal to zero (in other words, this is equivalent to '.if').

'.ifnes STRING1,STRING2'
  Like '.ifeqs', but the sense of the test is reversed: this
  assembles the following section of code if the two strings are not
  the same.

```



File: as.info, Node: Incbin, Next: Include, Prev: If, Up: Pseudo Ops

```
7.42 '.incbin "FILE"[,SKIP[,COUNT]]'
```

```
=====
```

The 'incbin' directive includes FILE verbatim at the current location. You can control the search paths used with the '-I' command-line option (*note Command-Line Options: Invoking.). Quotation marks are required around FILE.

The SKIP argument skips a number of bytes from the start of the FILE. The COUNT argument indicates the maximum number of bytes to read. Note that the data is not aligned in any way, so it is the user's

responsibility to make sure that proper alignment is provided both before and after the 'incbin' directive.



File: as.info, Node: Include, Next: Int, Prev: Incbin, Up: Pseudo Ops

7.43 '.include "FILE"'

=====

This directive provides a way to include supporting files at specified points in your source program. The code from FILE is assembled as if it followed the point of the '.include'; when the end of the included file is reached, assembly of the original file continues. You can control the search paths used with the '-I' command-line option (*note Command-Line Options: Invoking.). Quotation marks are required around FILE.



File: as.info, Node: Int, Next: Internal, Prev: Include, Up: Pseudo Ops

7.44 '.int EXPRESSIONS'

=====

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order and bit size of the number depends on what kind of target the assembly is for.



File: as.info, Node: Internal, Next: Irp, Prev: Int, Up: Pseudo Ops

7.45 '.internal NAMES'

=====

This is one of the ELF visibility directives. The other two are '.hidden' (*note '.hidden': Hidden.) and '.protected' (*note '.protected': Protected.).

This directive overrides the named symbols default visibility (which is set by their binding: local, global or weak). The directive sets the visibility to 'internal' which means that the symbols are considered to be 'hidden' (i.e., not visible to other components), and that some extra, processor specific processing must also be performed upon the symbols as well.



File: as.info, Node: Irp, Next: Irpc, Prev: Internal, Up: Pseudo Ops

7.46 '.irp SYMBOL,VALUES'...

=====

Evaluate a sequence of statements assigning different values to SYMBOL. The sequence of statements starts at the '.irp' directive, and is terminated by an '.endr' directive. For each VALUE, SYMBOL is set to VALUE, and the sequence of statements is assembled. If no VALUE is listed, the sequence of statements is assembled once, with SYMBOL set to the null string. To refer to SYMBOL within the sequence of statements, use \SYMBOL.

For example, assembling

```
.irp    param,1,2,3
move    d\param,sp@-
.endr
```

is equivalent to assembling

```
move    d1,sp@-
move    d2,sp@-
move    d3,sp@-
```

For some caveats with the spelling of SYMBOL, see also `*note Macro::`.

us

File: as.info, Node: Irpc, Next: Lcomm, Prev: Irp, Up: Pseudo Ops

7.47 '.irpc SYMBOL,VALUES'...

=====

Evaluate a sequence of statements assigning different values to SYMBOL. The sequence of statements starts at the '.irpc' directive, and is terminated by an '.endr' directive. For each character in VALUE, SYMBOL is set to the character, and the sequence of statements is assembled. If no VALUE is listed, the sequence of statements is assembled once, with SYMBOL set to the null string. To refer to SYMBOL within the sequence of statements, use \SYMBOL.

For example, assembling

```
.irpc    param,123
move    d\param,sp@-
.endr
```

is equivalent to assembling

```
move    d1,sp@-
move    d2,sp@-
move    d3,sp@-
```

For some caveats with the spelling of SYMBOL, see also the discussion at `*Note Macro::`.

us

File: as.info, Node: Lcomm, Next: Lflags, Prev: Irpc, Up: Pseudo Ops

7.48 '.lcomm SYMBOL , LENGTH'

=====

Reserve LENGTH (an absolute expression) bytes for a local common denoted by SYMBOL. The section and value of SYMBOL are those of the new local common. The addresses are allocated in the bss section, so that at run-time the bytes start off zeroed. SYMBOL is not declared global (`*note '.global': Global.`), so is normally not visible to 'ld'.

Some targets permit a third argument to be used with '.lcomm'. This argument specifies the desired alignment of the symbol in the bss section.

The syntax for `.lcomm` differs slightly on the HPPA. The syntax is `'SYMBOL .lcomm, LENGTH'`; `SYMBOL` is optional.

`US`

File: as.info, Node: Lflags, Next: Line, Prev: Lcomm, Up: Pseudo Ops

7.49 `.lflags`

=====

'as' accepts this directive, for compatibility with other assemblers, but ignores it.

`US`

File: as.info, Node: Line, Next: Linkonce, Prev: Lflags, Up: Pseudo Ops

7.50 `.line LINE-NUMBER`

=====

Change the logical line number. `LINE-NUMBER` must be an absolute expression. The next line has that logical line number. Therefore any other statements on the current line (after a statement separator character) are reported as on logical line number `LINE-NUMBER - 1`. One day 'as' will no longer support this directive: it is recognized only for compatibility with existing assembler programs.

Even though this is a directive associated with the `'a.out'` or `'b.out'` object-code formats, 'as' still recognizes it when producing COFF output, and treats `.line` as though it were the COFF `'.ln'` `_if_` it is found outside a `'.def'/'_endif'` pair.

Inside a `'.def'`, `.line` is, instead, one of the directives used by compilers to generate auxiliary symbol information for debugging.

`US`

File: as.info, Node: Linkonce, Next: List, Prev: Line, Up: Pseudo Ops

7.51 `.linkonce [TYPE]`

=====

Mark the current section so that the linker only includes a single copy of it. This may be used to include the same section in several different object files, but ensure that the linker will only include it once in the final output file. The `.linkonce` pseudo-op must be used for each instance of the section. Duplicate sections are detected based on the section name, so it should be unique.

This directive is only supported by a few object file formats; as of this writing, the only object file format which supports it is the Portable Executable format used on Windows NT.

The `TYPE` argument is optional. If specified, it must be one of the following strings. For example:

```
.linkonce same_size
```

Not all types may be supported on all object file formats.

`'discard'`

Silently discard duplicate sections. This is the default.

`'one_only'`

Warn if there are duplicate sections, but still keep only one copy.

'same_size'

Warn if any of the duplicates have different sizes.

'same_contents'

Warn if any of the duplicates do not have exactly the same contents.

US

File: as.info, Node: List, Next: Ln, Prev: Linkonce, Up: Pseudo Ops

7.52 '.list'

=====

Control (in conjunction with the '.nolist' directive) whether or not assembly listings are generated. These two directives maintain an internal counter (which is zero initially). '.list' increments the counter, and '.nolist' decrements it. Assembly listings are generated whenever the counter is greater than zero.

By default, listings are disabled. When you enable them (with the '-a' command line option; *note Command-Line Options: Invoking.), the initial value of the listing counter is one.

US

File: as.info, Node: Ln, Next: Loc, Prev: List, Up: Pseudo Ops

7.53 '.ln LINE-NUMBER'

=====

'.ln' is a synonym for '.line'.

US

File: as.info, Node: Loc, Next: Loc_mark_labels, Prev: Ln, Up: Pseudo Ops

7.54 '.loc FILENO LINENO [COLUMN] [OPTIONS]'

=====

When emitting DWARF2 line number information, the '.loc' directive will add a row to the '.debug_line' line number matrix corresponding to the immediately following assembly instruction. The FILENO, LINENO, and optional COLUMN arguments will be applied to the '.debug_line' state machine before the row is added.

The OPTIONS are a sequence of the following tokens in any order:

'basic_block'

This option will set the 'basic_block' register in the '.debug_line' state machine to 'true'.

'prologue_end'

This option will set the 'prologue_end' register in the '.debug_line' state machine to 'true'.

'epilogue_begin'

This option will set the 'epilogue_begin' register in the '.debug_line' state machine to 'true'.

'is_stmt VALUE'

This option will set the 'is_stmt' register in the '.debug_line' state machine to 'value', which must be either 0 or 1.

'isa VALUE'

This directive will set the 'isa' register in the '.debug_line' state machine to VALUE, which must be an unsigned integer.

'discriminator VALUE'

This directive will set the 'discriminator' register in the '.debug_line' state machine to VALUE, which must be an unsigned integer.



File: as.info, Node: Loc_mark_labels, Next: Local, Prev: Loc, Up: Pseudo Ops

7.55 '.loc_mark_labels ENABLE'

=====

When emitting DWARF2 line number information, the '.loc_mark_labels' directive makes the assembler emit an entry to the '.debug_line' line number matrix with the 'basic_block' register in the state machine set whenever a code label is seen. The ENABLE argument should be either 1 or 0, to enable or disable this function respectively.



File: as.info, Node: Local, Next: Long, Prev: Loc_mark_labels, Up: Pseudo Ops

7.56 '.local NAMES'

=====

This directive, which is available for ELF targets, marks each symbol in the comma-separated list of 'names' as a local symbol so that it will not be externally visible. If the symbols do not already exist, they will be created.

For targets where the '.lcomm' directive (*note Lcomm::) does not accept an alignment argument, which is the case for most ELF targets, the '.local' directive can be used in combination with '.comm' (*note Comm::) to define aligned local common data.



File: as.info, Node: Long, Next: Macro, Prev: Local, Up: Pseudo Ops

7.57 '.long EXPRESSIONS'

=====

'long' is the same as 'int'. *Note 'int': Int.



File: as.info, Node: Macro, Next: MRI, Prev: Long, Up: Pseudo Ops

7.58 '.macro'

=====

The commands '.macro' and '.endm' allow you to define macros that generate assembly output. For example, this definition specifies a macro 'sum' that puts a sequence of numbers into memory:

```
.macro sum from=0, to=5
.long \from
.if \to-\from
sum "(\from+1)",\to
.endif
.endm
```

With that definition, 'SUM 0,5' is equivalent to this assembly input:

```
.long 0
.long 1
.long 2
.long 3
.long 4
.long 5
```

```
'.macro MACNAME'
```

```
'.macro MACNAME MACARGS ...'
```

Begin the definition of a macro called MACNAME. If your macro definition requires arguments, specify their names after the macro name, separated by commas or spaces. You can qualify the macro argument to indicate whether all invocations must specify a non-blank value (through ':req'), or whether it takes all of the remaining arguments (through ':vararg'). You can supply a default value for any macro argument by following the name with '=DEFLT'. You cannot define two macros with the same MACNAME unless it has been subject to the '.purgem' directive (*note Purgem:.) between the two definitions. For example, these are all valid '.macro' statements:

```
'.macro comm'
```

Begin the definition of a macro called 'comm', which takes no arguments.

```
'.macro plus1 p, p1'
```

```
'.macro plus1 p p1'
```

Either statement begins the definition of a macro called 'plus1', which takes two arguments; within the macro definition, write '\p' or '\p1' to evaluate the arguments.

```
'.macro reserve_str p1=0 p2'
```

Begin the definition of a macro called 'reserve_str', with two arguments. The first argument has a default value, but not the second. After the definition is complete, you can call the macro either as 'reserve_str A,B' (with '\p1' evaluating to A and '\p2' evaluating to B), or as 'reserve_str ,B' (with '\p1' evaluating as the default, in this case '0', and '\p2' evaluating to B).

```
'.macro m p1:req, p2=0, p3:vararg'
```

Begin the definition of a macro called 'm', with at least three arguments. The first argument must always have a value specified, but not the second, which instead has a default value. The third formal will get assigned all remaining arguments specified at invocation time.

When you call a macro, you can specify the argument values either by position, or by keyword. For example, 'sum 9,17' is equivalent to 'sum to=17, from=9'.

Note that since each of the MACARGS can be an identifier exactly as any other one permitted by the target architecture, there may be occasional problems if the target hand-crafts special meanings to certain characters when they occur in a special position. For example, if the colon (':') is generally permitted to be part of a symbol name, but the architecture specific code special-cases it when occurring as the final character of a symbol (to denote a label), then the macro parameter replacement code will have no way of knowing that and consider the whole construct (including the colon) an identifier, and check only this identifier for being the subject to parameter substitution. So for example this macro definition:

```
.macro label l
\l:
.endm
```

might not work as expected. Invoking 'label foo' might not create a label called 'foo' but instead just insert the text '\l:' into the assembler source, probably generating an error about an unrecognised identifier.

Similarly problems might occur with the period character ('.') which is often allowed inside opcode names (and hence identifier names). So for example constructing a macro to build an opcode from a base name and a length specifier like this:

```
.macro opcode base length
    \base.\length
.endm
```

and invoking it as 'opcode store l' will not create a 'store.l' instruction but instead generate some kind of error as the assembler tries to interpret the text '\base.\length'.

There are several possible ways around this problem:

'Insert white space'

If it is possible to use white space characters then this is the simplest solution. eg:

```
.macro label l
\l :
.endm
```

'Use '\()''

The string '\()' can be used to separate the end of a macro argument from the following text. eg:

```
.macro opcode base length
    \base\().\length
.endm
```

'Use the alternate macro syntax mode'

In the alternative macro syntax mode the ampersand character ('&') can be used as a separator. eg:

```
.altmacro
```

```
.macro label l
l&:
.endm
```

Note: this problem of correctly identifying string parameters to pseudo ops also applies to the identifiers used in `'.irp'` (*note Irp::) and `'.irpc'` (*note Irpc::) as well.

```
'.endm'
Mark the end of a macro definition.
```

```
'.exitm'
Exit early from the current macro definition.
```

```
'\@'
'as' maintains a counter of how many macros it has executed in this
pseudo-variable; you can copy that number to your output with '@',
but only within a macro definition.
```

```
'LOCAL NAME [ , ... ]'
Warning: 'LOCAL' is only available if you select "alternate macro
syntax" with '--alternate' or '.altmacro'. *Note 'altmacro':
Altmacro.
```



File: as.info, Node: MRI, Next: Noaltmacro, Prev: Macro, Up: Pseudo Ops

```
7.59 '.mri VAL'
=====
```

If VAL is non-zero, this tells 'as' to enter MRI mode. If VAL is zero, this tells 'as' to exit MRI mode. This change affects code assembled until the next `'.mri'` directive, or until the end of the file. *Note MRI mode: M.



File: as.info, Node: Noaltmacro, Next: Nolist, Prev: MRI, Up: Pseudo Ops

```
7.60 '.noaltmacro'
=====
```

Disable alternate macro mode. *Note Altmacro::.



File: as.info, Node: Nolist, Next: Octa, Prev: Noaltmacro, Up: Pseudo Ops

```
7.61 '.nolist'
=====
```

Control (in conjunction with the `'.list'` directive) whether or not assembly listings are generated. These two directives maintain an internal counter (which is zero initially). `'.list'` increments the counter, and `'.nolist'` decrements it. Assembly listings are generated whenever the counter is greater than zero.



File: as.info, Node: Octa, Next: Offset, Prev: Nolist, Up: Pseudo Ops

```
7.62 '.octa BIGNUMS'
```

=====

This directive expects zero or more bignums, separated by commas. For each bignum, it emits a 16-byte integer.

The term "octa" comes from contexts in which a "word" is two bytes; hence `_octa_`-word for 16 bytes.

US

File: as.info, Node: Offset, Next: Org, Prev: Octa, Up: Pseudo Ops

7.63 `'.offset LOC'`

=====

Set the location counter to LOC in the absolute section. LOC must be an absolute expression. This directive may be useful for defining symbols with absolute values. Do not confuse it with the `'.org'` directive.

US

File: as.info, Node: Org, Next: P2align, Prev: Offset, Up: Pseudo Ops

7.64 `'.org NEW-LC , FILL'`

=====

Advance the location counter of the current section to NEW-LC. NEW-LC is either an absolute expression or an expression with the same section as the current subsection. That is, you can't use `'.org'` to cross sections: if NEW-LC has the wrong section, the `'.org'` directive is ignored. To be compatible with former assemblers, if the section of NEW-LC is absolute, 'as' issues a warning, then pretends the section of NEW-LC is the same as the current subsection.

`'.org'` may only increase the location counter, or leave it unchanged; you cannot use `'.org'` to move the location counter backwards.

Because 'as' tries to assemble programs in one pass, NEW-LC may not be undefined. If you really detest this restriction we eagerly await a chance to share your improved assembler.

Beware that the origin is relative to the start of the section, not to the start of the subsection. This is compatible with other people's assemblers.

When the location counter (of the current subsection) is advanced, the intervening bytes are filled with FILL which should be an absolute expression. If the comma and FILL are omitted, FILL defaults to zero.

US

File: as.info, Node: P2align, Next: PopSection, Prev: Org, Up: Pseudo Ops

7.65 `'.p2align[wl] ABS-EXPR, ABS-EXPR, ABS-EXPR'`

=====

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the number of low-order zero bits the location counter must have after advancement. For example `'.p2align 3'` advances the location counter until it a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

The second expression (also absolute) gives the fill value to be stored in the padding bytes. It (and the comma) may be omitted. If it is omitted, the padding bytes are normally zero. However, on some systems, if the section is marked as containing code and the fill value is omitted, the space is filled with no-op instructions.

The third expression is also absolute, and is also optional. If it is present, it is the maximum number of bytes that should be skipped by this alignment directive. If doing the alignment would require skipping more bytes than the specified maximum, then the alignment is not done at all. You can omit the fill value (the second argument) entirely by simply using two commas after the required alignment; this can be useful if you want the alignment to be filled with no-op instructions when appropriate.

The `.p2alignw` and `.p2alignl` directives are variants of the `.p2align` directive. The `.p2alignw` directive treats the fill pattern as a two byte word value. The `.p2alignl` directive treats the fill pattern as a four byte longword value. For example, `.p2alignw 2,0x368d` will align to a multiple of 4. If it skips two bytes, they will be filled in with the value 0x368d (the exact placement of the bytes depends upon the endianness of the processor). If it skips 1 or 3 bytes, the fill value is undefined.

US

File: as.info, Node: PopSection, Next: Previous, Prev: P2align, Up: Pseudo Ops

7.66 `.popsection`

=====

This is one of the ELF section stack manipulation directives. The others are `.section` (`*note Section::`), `.subsection` (`*note SubSection::`), `.pushsection` (`*note PushSection::`), and `.previous` (`*note Previous::`).

This directive replaces the current section (and subsection) with the top section (and subsection) on the section stack. This section is popped off the stack.

US

File: as.info, Node: Previous, Next: Print, Prev: PopSection, Up: Pseudo Ops

7.67 `.previous`

=====

This is one of the ELF section stack manipulation directives. The others are `.section` (`*note Section::`), `.subsection` (`*note SubSection::`), `.pushsection` (`*note PushSection::`), and `.popsection` (`*note PopSection::`).

This directive swaps the current section (and subsection) with most recently referenced section/subsection pair prior to this one. Multiple `.previous` directives in a row will flip between two sections (and their subsections). For example:

```
.section A
  .subsection 1
    .word 0x1234
```

```
.subsection 2
.word 0x5678
.previous
.word 0x9abc
```

Will place 0x1234 and 0x9abc into subsection 1 and 0x5678 into subsection 2 of section A. Whilst:

```
.section A
.subsection 1
# Now in section A subsection 1
.word 0x1234
.section B
.subsection 0
# Now in section B subsection 0
.word 0x5678
.subsection 1
# Now in section B subsection 1
.word 0x9abc
.previous
# Now in section B subsection 0
.word 0xdef0
```

Will place 0x1234 into section A, 0x5678 and 0xdef0 into subsection 0 of section B and 0x9abc into subsection 1 of section B.

In terms of the section stack, this directive swaps the current section with the top section on the section stack.

US

File: as.info, Node: Print, Next: Protected, Prev: Previous, Up: Pseudo Ops

7.68 '.print STRING'

=====

'as' will print STRING on the standard output during assembly. You must put STRING in double quotes.

US

File: as.info, Node: Protected, Next: Psize, Prev: Print, Up: Pseudo Ops

7.69 '.protected NAMES'

=====

This is one of the ELF visibility directives. The other two are '.hidden' (*note Hidden::) and '.internal' (*note Internal::).

This directive overrides the named symbols default visibility (which is set by their binding: local, global or weak). The directive sets the visibility to 'protected' which means that any references to the symbols from within the components that defines them must be resolved to the definition in that component, even if a definition in another component would normally preempt this.

US

File: as.info, Node: Psize, Next: Purgem, Prev: Protected, Up: Pseudo Ops

7.70 '.psize LINES , COLUMNS'

=====

Use this directive to declare the number of lines--and, optionally, the number of columns--to use for each page, when generating listings.

If you do not use `'.psize'`, listings use a default line-count of 60. You may omit the comma and `COLUMNS` specification; the default width is 200 columns.

`'as'` generates formfeeds whenever the specified number of lines is exceeded (or whenever you explicitly request one, using `'.eject'`).

If you specify `LINES` as `'0'`, no formfeeds are generated save those explicitly specified with `'.eject'`.

US

File: as.info, Node: Purgem, Next: PushSection, Prev: Psize, Up: Pseudo Ops

7.71 `'.purgem NAME'`

=====

Undefine the macro `NAME`, so that later uses of the string will not be expanded. `*Note Macro::`.

US

File: as.info, Node: PushSection, Next: Quad, Prev: Purgem, Up: Pseudo Ops

7.72 `'.pushsection NAME [, SUBSECTION] [, "FLAGS"[, @TYPE[, ARGUMENTS]]]'`

=====

This is one of the ELF section stack manipulation directives. The others are `'.section'` (`*note Section::`), `'.subsection'` (`*note SubSection::`), `'.popsection'` (`*note PopSection::`), and `'.previous'` (`*note Previous::`).

This directive pushes the current section (and subsection) onto the top of the section stack, and then replaces the current section and subsection with `'name'` and `'subsection'`. The optional `'flags'`, `'type'` and `'arguments'` are treated the same as in the `'.section'` (`*note Section::`) directive.

US

File: as.info, Node: Quad, Next: Reloc, Prev: PushSection, Up: Pseudo Ops

7.73 `'.quad BIGNUMS'`

=====

`'.quad'` expects zero or more bignums, separated by commas. For each bignum, it emits an 8-byte integer. If the bignum won't fit in 8 bytes, it prints a warning message; and just takes the lowest order 8 bytes of the bignum.

The term "quad" comes from contexts in which a "word" is two bytes; hence `_quad_`-word for 8 bytes.

US

File: as.info, Node: Reloc, Next: Rept, Prev: Quad, Up: Pseudo Ops

7.74 `'.reloc OFFSET, RELOC_NAME[, EXPRESSION]'`

=====

Generate a relocation at OFFSET of type RELOC_NAME with value EXPRESSION. If OFFSET is a number, the relocation is generated in the current section. If OFFSET is an expression that resolves to a symbol plus offset, the relocation is generated in the given symbol's section. EXPRESSION, if present, must resolve to a symbol plus addend or to an absolute value, but note that not all targets support an addend. e.g. ELF REL targets such as i386 store an addend in the section contents rather than in the relocation. This low level interface does not support addends stored in the section.



File: as.info, Node: Rept, Next: Sbttl, Prev: Reloc, Up: Pseudo Ops

7.75 '.rept COUNT'

=====

Repeat the sequence of lines between the '.rept' directive and the next '.endr' directive COUNT times.

For example, assembling

```
.rept 3
.long 0
.endr
```

is equivalent to assembling

```
.long 0
.long 0
.long 0
```



File: as.info, Node: Sbttl, Next: Scl, Prev: Rept, Up: Pseudo Ops

7.76 '.sbttl "SUBHEADING"'

=====

Use SUBHEADING as the title (third line, immediately after the title line) when generating assembly listings.

This directive affects subsequent pages, as well as the current page if it appears within ten lines of the top of a page.



File: as.info, Node: Scl, Next: Section, Prev: Sbttl, Up: Pseudo Ops

7.77 '.scl CLASS'

=====

Set the storage-class value for a symbol. This directive may only be used inside a '.def'/'.'endif' pair. Storage class may flag whether a symbol is static or external, or it may record further symbolic debugging information.



File: as.info, Node: Section, Next: Set, Prev: Scl, Up: Pseudo Ops

7.78 '.section NAME'

=====

Use the `'.section'` directive to assemble the following code into a section named NAME.

This directive is only supported for targets that actually support arbitrarily named sections; on `'a.out'` targets, for example, it is not accepted, even with a standard `'a.out'` section name.

COFF Version

For COFF targets, the `'.section'` directive is used in one of the following ways:

```
.section NAME[, "FLAGS"]
.section NAME[, SUBSECTION]
```

If the optional argument is quoted, it is taken as flags to use for the section. Each flag is a single character. The following flags are recognized:

```
'b'      bss section (uninitialized data)
'n'      section is not loaded
'w'      writable section
'd'      data section
'e'      exclude section from linking
'r'      read-only section
'x'      executable section
's'      shared section (meaningful for PE targets)
'a'      ignored. (For compatibility with the ELF version)
'y'      section is not readable (meaningful for PE targets)
'0-9'    single-digit power-of-two section alignment (GNU extension)
```

If no flags are specified, the default flags depend upon the section name. If the section name is not recognized, the default will be for the section to be loaded and writable. Note the `'n'` and `'w'` flags remove attributes from the section, rather than adding them, so if they are used on their own it will be as if no flags had been specified at all.

If the optional argument to the `'.section'` directive is not quoted, it is taken as a subsection number (*note Sub-Sections:).

ELF Version

This is one of the ELF section stack manipulation directives. The

others are `'.subsection'` (`*note SubSection::`), `'.pushsection'` (`*note PushSection::`), `'.popsection'` (`*note PopSection::`), and `'.previous'` (`*note Previous::`).

For ELF targets, the `'.section'` directive is used like this:

```
.section NAME [, "FLAGS"[, @TYPE[,FLAG_SPECIFIC_ARGUMENTS]]]
```

If the `'--sectname-subst'` command-line option is provided, the `NAME` argument may contain a substitution sequence. Only `'%S'` is supported at the moment, and substitutes the current section name. For example:

```
.macro exception_code
.section %S.exception
[exception code here]
.previous
.endm
```

```
.text
[code]
exception_code
[...]
```

```
.section .init
[init code]
exception_code
[...]
```

The two `'exception_code'` invocations above would create the `'.text.exception'` and `'.init.exception'` sections respectively. This is useful e.g. to discriminate between ancillary sections that are tied to setup code to be discarded after use from ancillary sections that need to stay resident without having to define multiple `'exception_code'` macros just for that purpose.

The optional `FLAGS` argument is a quoted string which may contain any combination of the following characters:

```
'a'    section is allocatable
'e'    section is excluded from executable and shared library.
'w'    section is writable
'x'    section is executable
'M'    section is mergeable
'S'    section contains zero terminated strings
'G'    section is a member of a section group
'T'    section is used for thread-local-storage
'?'    section is a member of the previously-current section's group, if
any
'<number>'
a numeric value indicating the bits to be set in the ELF section
```

header's flags field. Note - if one or more of the alphabetic characters described above is also included in the flags field, their bit values will be ORed into the resulting value.

'<target specific>'

some targets extend this list with their own flag characters

Note - once a section's flags have been set they cannot be changed. There are a few exceptions to this rule however. Processor and application specific flags can be added to an already defined section. The '.interp', '.strtab' and '.symtab' sections can have the allocate flag ('a') set after they are initially defined, and the '.note-GNU-stack' section may have the executable ('x') flag added.

The optional TYPE argument may contain one of the following constants:

'@progbits'

section contains data

'@nobits'

section does not contain data (i.e., section only occupies space)

'@note'

section contains data which is used by things other than the program

'@init_array'

section contains an array of pointers to init functions

'@fini_array'

section contains an array of pointers to finish functions

'@preinit_array'

section contains an array of pointers to pre-init functions

'@<number>'

a numeric value to be set as the ELF section header's type field.

'@<target specific>'

some targets extend this list with their own types

Many targets only support the first three section types. The type may be enclosed in double quotes if necessary.

Note on targets where the '@' character is the start of a comment (eg ARM) then another character is used instead. For example the ARM port uses the '%' character.

Note - some sections, eg '.text' and '.data' are considered to be special and have fixed types. Any attempt to declare them with a different type will generate an error from the assembler.

If FLAGS contains the 'M' symbol then the TYPE argument must be specified as well as an extra argument--ENTSIZE--like this:

```
.section NAME , "FLAGS"M, @TYPE, ENTSIZE
```

Sections with the 'M' flag but not 'S' flag must contain fixed size constants, each ENTSIZE octets long. Sections with both 'M' and 'S' must contain zero terminated strings where each character is ENTSIZE bytes long. The linker may remove duplicates within sections with the same name, same entity size and same flags. ENTSIZE must be an absolute expression. For sections with both 'M' and 'S', a string which is a suffix of a larger string is considered a duplicate. Thus '"def"' will be merged with '"abcdef"'; A reference to the first '"def"' will be changed to a reference to '"abcdef"+3'.

If FLAGS contains the 'G' symbol then the TYPE argument must be present along with an additional field like this:

```
.section NAME , "FLAGS"G, @TYPE, GROUPNAME[, LINKAGE]
```

The GROUPNAME field specifies the name of the section group to which this particular section belongs. The optional linkage field can contain:

```
'comdat'
    indicates that only one copy of this section should be retained
'.gnu.linkonce'
    an alias for comdat
```

Note: if both the M and G flags are present then the fields for the Merge flag should come first, like this:

```
.section NAME , "FLAGS"MG, @TYPE, ENTSIZE, GROUPNAME[, LINKAGE]
```

If FLAGS contains the '?' symbol then it may not also contain the 'G' symbol and the GROUPNAME or LINKAGE fields should not be present. Instead, '?' says to consider the section that's current before this directive. If that section used 'G', then the new section will use 'G' with those same GROUPNAME and LINKAGE fields implicitly. If not, then the '?' symbol has no effect.

If no flags are specified, the default flags depend upon the section name. If the section name is not recognized, the default will be for the section to have none of the above flags: it will not be allocated in memory, nor writable, nor executable. The section will contain data.

For ELF targets, the assembler supports another type of '.section' directive for compatibility with the Solaris assembler:

```
.section "NAME"[, FLAGS...]
```

Note that the section name is quoted. There may be a sequence of comma separated flags:

```
'#alloc'
    section is allocatable
'#write'
    section is writable
'#execinstr'
    section is executable
'#exclude'
    section is excluded from executable and shared library.
'#tls'
    section is used for thread local storage
```

This directive replaces the current section and subsection. See the contents of the gas test suite directory 'gas/testsuite/gas/elf' for some examples of how this directive and the other section stack directives work.

7.79 '.set SYMBOL, EXPRESSION'

=====

Set the value of SYMBOL to EXPRESSION. This changes SYMBOL's value and type to conform to EXPRESSION. If SYMBOL was flagged as external, it remains flagged (*note Symbol Attributes:).

You may '.set' a symbol many times in the same assembly provided that the values given to the symbol are constants. Values that are based on expressions involving other symbols are allowed, but some targets may restrict this to only being done once per assembly. This is because those targets do not set the addresses of symbols at assembly time, but rather delay the assignment until a final link is performed. This allows the linker a chance to change the code in the files, changing the location of, and the relative distance between, various different symbols.

If you '.set' a global symbol, the value stored in the object file is the last value stored into it.

On Z80 'set' is a real instruction, use 'SYMBOL defl EXPRESSION' instead.



File: as.info, Node: Short, Next: Single, Prev: Set, Up: Pseudo Ops

7.80 '.short EXPRESSIONS'

=====

'.short' is normally the same as '.word'. *Note '.word': Word.

In some configurations, however, '.short' and '.word' generate numbers of different lengths. *Note Machine Dependencies:.



File: as.info, Node: Single, Next: Size, Prev: Short, Up: Pseudo Ops

7.81 '.single FLONUMS'

=====

This directive assembles zero or more flonums, separated by commas. It has the same effect as '.float'. The exact kind of floating point numbers emitted depends on how 'as' is configured. *Note Machine Dependencies:.



File: as.info, Node: Size, Next: Skip, Prev: Single, Up: Pseudo Ops

7.82 '.size'

=====

This directive is used to set the size associated with a symbol.

COFF Version

For COFF targets, the '.size' directive is only permitted inside '.def'/'.'endef' pairs. It is used like this:

`.size` EXPRESSION

ELF Version

For ELF targets, the `.size` directive is used like this:

`.size` NAME , EXPRESSION

This directive sets the size associated with a symbol NAME. The size in bytes is computed from EXPRESSION which can make use of label arithmetic. This directive is typically used to set the size of function symbols.

US

File: as.info, Node: Skip, Next: Sleb128, Prev: Size, Up: Pseudo Ops

7.83 `.skip` SIZE , FILL'

=====

This directive emits SIZE bytes, each of value FILL. Both SIZE and FILL are absolute expressions. If the comma and FILL are omitted, FILL is assumed to be zero. This is the same as `.space`.

US

File: as.info, Node: Sleb128, Next: Space, Prev: Skip, Up: Pseudo Ops

7.84 `.sleb128` EXPRESSIONS'

=====

SLEB128 stands for "signed little endian base 128." This is a compact, variable length representation of numbers used by the DWARF symbolic debugging format. *Note `.uleb128`: Uleb128.

US

File: as.info, Node: Space, Next: Stab, Prev: Sleb128, Up: Pseudo Ops

7.85 `.space` SIZE , FILL'

=====

This directive emits SIZE bytes, each of value FILL. Both SIZE and FILL are absolute expressions. If the comma and FILL are omitted, FILL is assumed to be zero. This is the same as `.skip`.

Warning: `.space` has a completely different meaning for HPPA targets; use `.block` as a substitute. See 'HP9000 Series 800 Assembly Language Reference Manual' (HP 92432-90001) for the meaning of the `.space` directive. *Note HPPA Assembler Directives: HPPA Directives, for a summary.

US

File: as.info, Node: Stab, Next: String, Prev: Space, Up: Pseudo Ops

7.86 `.stabd`, `.stabn`, `.stabs`'

=====

There are three directives that begin `.stab`. All emit symbols (*note Symbols:), for use by symbolic debuggers. The symbols are not entered in the 'as' hash table: they cannot be referenced elsewhere in the

source file. Up to five fields are required:

STRING

This is the symbol's name. It may contain any character except '\000', so is more general than ordinary symbol names. Some debuggers used to code arbitrarily complex structures into symbol names using this field.

TYPE

An absolute expression. The symbol's type is set to the low 8 bits of this expression. Any bit pattern is permitted, but 'ld' and debuggers choke on silly bit patterns.

OTHER

An absolute expression. The symbol's "other" attribute is set to the low 8 bits of this expression.

DESC

An absolute expression. The symbol's descriptor is set to the low 16 bits of this expression.

VALUE

An absolute expression which becomes the symbol's value.

If a warning is detected while reading a '.stabd', '.stabn', or '.stabs' statement, the symbol has probably already been created; you get a half-formed symbol in your object file. This is compatible with earlier assemblers!

```
'.stabd TYPE , OTHER , DESC'
```

The "name" of the symbol generated is not even an empty string. It is a null pointer, for compatibility. Older assemblers used a null pointer so they didn't waste space in object files with empty strings.

The symbol's value is set to the location counter, relocatably. When your program is linked, the value of this symbol is the address of the location counter when the '.stabd' was assembled.

```
'.stabn TYPE , OTHER , DESC , VALUE'
```

The name of the symbol is set to the empty string ''.

```
'.stabs STRING , TYPE , OTHER , DESC , VALUE'
```

All five fields are specified.



File: as.info, Node: String, Next: Struct, Prev: Stab, Up: Pseudo Ops

```
7.87 '.string' "STR", '.string8' "STR", '.string16'
```

```
=====
```

```
"STR", '.string32' "STR", '.string64' "STR"
```

Copy the characters in STR to the object file. You may specify more than one string to copy, separated by commas. Unless otherwise specified for a particular machine, the assembler marks the end of each string with a 0 byte. You can use any of the escape sequences described in *note Strings: Strings.

The variants 'string16', 'string32' and 'string64' differ from the 'string' pseudo opcode in that each 8-bit character from STR is copied and expanded to 16, 32 or 64 bits respectively. The expanded characters are stored in target endianness byte order.

Example:

```
.string32 "BYE"
expands to:
.string    "\0\0\0Y\0\0\0E\0\0\0" /* On little endian targets. */
.string    "\0\0\0B\0\0\0Y\0\0\0E" /* On big endian targets. */
```



File: as.info, Node: Struct, Next: SubSection, Prev: String, Up: Pseudo Ops

7.88 '.struct EXPRESSION'

=====

Switch to the absolute section, and set the section offset to EXPRESSION, which must be an absolute expression. You might use this as follows:

```
.struct 0
field1:
.struct field1 + 4
field2:
.struct field2 + 4
field3:
```

This would define the symbol 'field1' to have the value 0, the symbol 'field2' to have the value 4, and the symbol 'field3' to have the value 8. Assembly would be left in the absolute section, and you would need to use a '.section' directive of some sort to change to some other section before further assembly.



File: as.info, Node: SubSection, Next: Symver, Prev: Struct, Up: Pseudo Ops

7.89 '.subsection NAME'

=====

This is one of the ELF section stack manipulation directives. The others are '.section' (*note Section:~), '.pushsection' (*note PushSection:~), '.popsection' (*note PopSection:~), and '.previous' (*note Previous:~).

This directive replaces the current subsection with 'name'. The current section is not changed. The replaced subsection is put onto the section stack in place of the then current top of stack subsection.



File: as.info, Node: Symver, Next: Tag, Prev: SubSection, Up: Pseudo Ops

7.90 '.symver'

=====

Use the '.symver' directive to bind symbols to specific version nodes within a source file. This is only supported on ELF platforms, and is typically used when assembling files to be linked into a shared library. There are cases where it may make sense to use this in objects to be bound into an application itself so as to override a versioned symbol

from a shared library.

For ELF targets, the `'.symver'` directive can be used like this:

```
.symver NAME, NAME2@NODENAME
```

If the symbol `NAME` is defined within the file being assembled, the `'.symver'` directive effectively creates a symbol alias with the name `NAME2@NODENAME`, and in fact the main reason that we just don't try and create a regular alias is that the `@` character isn't permitted in symbol names. The `NAME2` part of the name is the actual name of the symbol by which it will be externally referenced. The name `NAME` itself is merely a name of convenience that is used so that it is possible to have definitions for multiple versions of a function within a single source file, and so that the compiler can unambiguously know which version of a function is being mentioned. The `NODENAME` portion of the alias should be the name of a node specified in the version script supplied to the linker when building a shared library. If you are attempting to override a versioned symbol from a shared library, then `NODENAME` should correspond to the nodename of the symbol you are trying to override.

If the symbol `NAME` is not defined within the file being assembled, all references to `NAME` will be changed to `NAME2@NODENAME`. If no reference to `NAME` is made, `NAME2@NODENAME` will be removed from the symbol table.

Another usage of the `'.symver'` directive is:

```
.symver NAME, NAME2@@NODENAME
```

In this case, the symbol `NAME` must exist and be defined within the file being assembled. It is similar to `NAME2@NODENAME`. The difference is `NAME2@@NODENAME` will also be used to resolve references to `NAME2` by the linker.

The third usage of the `'.symver'` directive is:

```
.symver NAME, NAME2@@@NODENAME
```

When `NAME` is not defined within the file being assembled, it is treated as `NAME2@NODENAME`. When `NAME` is defined within the file being assembled, the symbol name, `NAME`, will be changed to `NAME2@@@NODENAME`.

US

File: as.info, Node: Tag, Next: Text, Prev: Symver, Up: Pseudo Ops

7.91 `'.tag STRUCTNAME'`

=====

This directive is generated by compilers to include auxiliary debugging information in the symbol table. It is only permitted inside `'.def'/''.endef'` pairs. Tags are used to link structure definitions in the symbol table with instances of those structures.

US

File: as.info, Node: Text, Next: Title, Prev: Tag, Up: Pseudo Ops

7.92 `'.text SUBSECTION'`

=====

Tells `'as'` to assemble the following statements onto the end of the text subsection numbered `SUBSECTION`, which is an absolute expression. If `SUBSECTION` is omitted, subsection number zero is used.

US

File: as.info, Node: Title, Next: Type, Prev: Text, Up: Pseudo Ops

7.93 `.title "HEADING"`

=====

Use HEADING as the title (second line, immediately after the source file name and pagenumber) when generating assembly listings.

This directive affects subsequent pages, as well as the current page if it appears within ten lines of the top of a page.



File: as.info, Node: Type, Next: Uleb128, Prev: Title, Up: Pseudo Ops

7.94 `.type'`

=====

This directive is used to set the type of a symbol.

COFF Version

For COFF targets, this directive is permitted only within `'.def'/' .endif'` pairs. It is used like this:

```
.type INT
```

This records the integer INT as the type attribute of a symbol table entry.

ELF Version

For ELF targets, the `.type'` directive is used like this:

```
.type NAME , TYPE DESCRIPTION
```

This sets the type of symbol NAME to be either a function symbol or an object symbol. There are five different syntaxes supported for the TYPE DESCRIPTION field, in order to provide compatibility with various other assemblers.

Because some of the characters used in these syntaxes (such as '@' and '#') are comment characters for some architectures, some of the syntaxes below do not work on all architectures. The first variant will be accepted by the GNU assembler on all architectures so that variant should be used for maximum portability, if you do not need to assemble your code with other assemblers.

The syntaxes supported are:

```
.type <name> STT_<TYPE_IN_UPPER_CASE>
.type <name> ,#<type>
.type <name> ,@<type>
.type <name> ,%<type>
.type <name> ,"<type>"
```

The types supported are:

'STT_FUNC'

'function'

Mark the symbol as being a function name.

'STT_GNU_IFUNC'

'gnu_indirect_function'

Mark the symbol as an indirect function when evaluated during reloc processing. (This is only supported on assemblers targeting GNU systems).

'STT_OBJECT'

'object'

Mark the symbol as being a data object.

'STT_TLS'

'tls_object'

Mark the symbol as being a thread-local data object.

'STT_COMMON'

'common'

Mark the symbol as being a common data object.

'STT_NOTYPE'

'notype'

Does not mark the symbol in any way. It is supported just for completeness.

'gnu_unique_object'

Marks the symbol as being 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. (This is only supported on assemblers targeting GNU systems).

Note: Some targets support extra types in addition to those listed above.



File: as.info, Node: Uleb128, Next: Val, Prev: Type, Up: Pseudo Ops

7.95 '.uleb128 EXPRESSIONS'

=====

ULEB128 stands for "unsigned little endian base 128." This is a compact, variable length representation of numbers used by the DWARF symbolic debugging format. *Note '.sleb128': Sleb128.



File: as.info, Node: Val, Next: Version, Prev: Uleb128, Up: Pseudo Ops

7.96 '.val ADDR'

=====

This directive, permitted only within '.def'/' .endef' pairs, records the address ADDR as the value attribute of a symbol table entry.



File: as.info, Node: Version, Next: VTableEntry, Prev: Val, Up: Pseudo Ops

7.97 '.version "STRING"'

=====

This directive creates a '.note' section and places into it an ELF formatted note of type NT_VERSION. The note's name is set to 'string'.



File: as.info, Node: VTableEntry, Next: VTableInherit, Prev: Version, Up: Pseudo Ops

7.98 '.vtable_entry TABLE, OFFSET'

=====

This directive finds or creates a symbol 'table' and creates a 'VTABLE_ENTRY' relocation for it with an addend of 'offset'.



File: as.info, Node: VTableInherit, Next: Warning, Prev: VTableEntry, Up: Pseudo Ops

7.99 '.vtable_inherit CHILD, PARENT'

=====

This directive finds the symbol 'child' and finds or creates the symbol 'parent' and then creates a 'VTABLE_INHERIT' relocation for the parent whose addend is the value of the child symbol. As a special case the parent name of '0' is treated as referring to the '*ABS*' section.



File: as.info, Node: Warning, Next: Weak, Prev: VTableInherit, Up: Pseudo Ops

7.100 '.warning "STRING"'

=====

Similar to the directive '.error' (*note '.error "STRING": Error.'), but just emits a warning.



File: as.info, Node: Weak, Next: Weakref, Prev: Warning, Up: Pseudo Ops

7.101 '.weak NAMES'

=====

This directive sets the weak attribute on the comma separated list of symbol 'names'. If the symbols do not already exist, they will be created.

On COFF targets other than PE, weak symbols are a GNU extension. This directive sets the weak attribute on the comma separated list of symbol 'names'. If the symbols do not already exist, they will be created.

On the PE target, weak symbols are supported natively as weak aliases. When a weak symbol is created that is not an alias, GAS creates an alternate symbol to hold the default value.



File: as.info, Node: Weakref, Next: Word, Prev: Weak, Up: Pseudo Ops

7.102 '.weakref ALIAS, TARGET'

=====

This directive creates an alias to the target symbol that enables the symbol to be referenced with weak-symbol semantics, but without actually making it weak. If direct references or definitions of the symbol are present, then the symbol will not be weak, but if all references to it are through weak references, the symbol will be marked as weak in the symbol table.

The effect is equivalent to moving all references to the alias to a separate assembly source file, renaming the alias to the symbol in it, declaring the symbol as weak there, and running a reloadable link to merge the object files resulting from the assembly of the new source file and the old source file that had the references to the alias removed.

The alias itself never makes to the symbol table, and is entirely handled within the assembler.



File: as.info, Node: Word, Next: Zero, Prev: Weakref, Up: Pseudo Ops

7.103 '.word EXPRESSIONS'

=====

This directive expects zero or more EXPRESSIONS, of any section, separated by commas.

The size of the number emitted, and its byte order, depend on what target computer the assembly is for.

Warning: Special Treatment to support Compilers

Machines with a 32-bit address space, but that do less than 32-bit addressing, require the following special treatment. If the machine of interest to you does 32-bit addressing (or doesn't require it; *note Machine Dependencies:), you can ignore this issue.

In order to assemble compiler output into something that works, 'as' occasionally does strange things to '.word' directives. Directives of the form '.word sym1-sym2' are often emitted by compilers as part of jump tables. Therefore, when 'as' assembles a directive of the form '.word sym1-sym2', and the difference between 'sym1' and 'sym2' does not fit in 16 bits, 'as' creates a "secondary jump table", immediately before the next label. This secondary jump table is preceded by a short-jump to the first byte after the secondary table. This short-jump prevents the flow of control from accidentally falling into the new table. Inside the table is a long-jump to 'sym2'. The original '.word' contains 'sym1' minus the address of the long-jump to 'sym2'.

If there were several occurrences of '.word sym1-sym2' before the secondary jump table, all of them are adjusted. If there was a '.word sym3-sym4', that also did not fit in sixteen bits, a long-jump to 'sym4' is included in the secondary jump table, and the '.word' directives are adjusted to contain 'sym3' minus the address of the long-jump to 'sym4'; and so on, for as many entries in the original jump table as necessary.



File: as.info, Node: Zero, Next: Deprecated, Prev: Word, Up: Pseudo Ops

7.104 '.zero SIZE'

=====

This directive emits SIZE 0-valued bytes. SIZE must be an absolute expression. This directive is actually an alias for the '.skip' directive so it can take an optional second argument of the value to store in the bytes instead of zero. Using '.zero' in this way would be confusing however.



File: as.info, Node: Deprecated, Prev: Zero, Up: Pseudo Ops

7.105 Deprecated Directives

=====

One day these directives won't work. They are included for compatibility with older assemblers.

.abort
.line



File: as.info, Node: Object Attributes, Next: Machine Dependencies, Prev: Pseudo Ops, Up: Top

8 Object Attributes

'as' assembles source files written for a specific architecture into object files for that architecture. But not all object files are alike. Many architectures support incompatible variations. For instance, floating point arguments might be passed in floating point registers if the object file requires hardware floating point support--or floating point arguments might be passed in integer registers if the object file supports processors with no hardware floating point unit. Or, if two objects are built for different generations of the same architecture, the combination may require the newer generation at run-time.

This information is useful during and after linking. At link time, 'ld' can warn about incompatible object files. After link time, tools like 'gdb' can use it to process the linked file correctly.

Compatibility information is recorded as a series of object attributes. Each attribute has a "vendor", "tag", and "value". The vendor is a string, and indicates who sets the meaning of the tag. The tag is an integer, and indicates what property the attribute describes. The value may be a string or an integer, and indicates how the property affects this object. Missing attributes are the same as attributes with a zero value or empty string value.

Object attributes were developed as part of the ABI for the ARM Architecture. The file format is documented in 'ELF for the ARM Architecture'.

* Menu:

* GNU Object Attributes:: GNU Object Attributes
* Defining New Object Attributes:: Defining New Object Attributes



File: as.info, Node: GNU Object Attributes, Next: Defining New Object Attributes,
Up: Object Attributes

8.1 GNU Object Attributes

=====

The `'.gnu_attribute'` directive records an object attribute with vendor `'gnu'`.

Except for `'Tag_compatibility'`, which has both an integer and a string for its value, GNU attributes have a string value if the tag number is odd and an integer value if the tag number is even. The second bit (`'TAG & 2'` is set for architecture-independent attributes and clear for architecture-dependent ones.

8.1.1 Common GNU attributes

These attributes are valid on all architectures.

Tag_compatibility (32)

The compatibility attribute takes an integer flag value and a vendor name. If the flag value is 0, the file is compatible with other toolchains. If it is 1, then the file is only compatible with the named toolchain. If it is greater than 1, the file can only be processed by other toolchains under some private arrangement indicated by the flag value and the vendor name.

8.1.2 MIPS Attributes

Tag_GNU_MIPS_ABI_FP (4)

The floating-point ABI used by this object file. The value will be:

- * 0 for files not affected by the floating-point ABI.
- * 1 for files using the hardware floating-point ABI with a standard double-precision FPU.
- * 2 for files using the hardware floating-point ABI with a single-precision FPU.
- * 3 for files using the software floating-point ABI.
- * 4 for files using the deprecated hardware floating-point ABI which used 64-bit floating-point registers, 32-bit general-purpose registers and increased the number of callee-saved floating-point registers.
- * 5 for files using the hardware floating-point ABI with a double-precision FPU with either 32-bit or 64-bit floating-point registers and 32-bit general-purpose registers.
- * 6 for files using the hardware floating-point ABI with 64-bit floating-point registers and 32-bit general-purpose registers.
- * 7 for files using the hardware floating-point ABI with 64-bit floating-point registers, 32-bit general-purpose registers and a rule that forbids the direct use of odd-numbered single-precision floating-point registers.

8.1.3 PowerPC Attributes

Tag_GNU_Power_ABI_FP (4)

The floating-point ABI used by this object file. The value will be:

- * 0 for files not affected by the floating-point ABI.
- * 1 for files using double-precision hardware floating-point ABI.
- * 2 for files using the software floating-point ABI.
- * 3 for files using single-precision hardware floating-point ABI.

Tag_GNU_Power_ABI_Vector (8)

The vector ABI used by this object file. The value will be:

- * 0 for files not affected by the vector ABI.
- * 1 for files using general purpose registers to pass vectors.
- * 2 for files using AltiVec registers to pass vectors.
- * 3 for files using SPE registers to pass vectors.

8.1.4 IBM z Systems Attributes

Tag_GNU_S390_ABI_Vector (8)

The vector ABI used by this object file. The value will be:

- * 0 for files not affected by the vector ABI.
- * 1 for files using software vector ABI.
- * 2 for files using hardware vector ABI.



File: as.info, Node: Defining New Object Attributes, Prev: GNU Object Attributes, Up: Object Attributes

8.2 Defining New Object Attributes

=====

If you want to define a new GNU object attribute, here are the places you will need to modify. New attributes should be discussed on the 'binutils' mailing list.

- * This manual, which is the official register of attributes.
- * The header for your architecture 'include/elf', to define the tag.
- * The 'bfd' support file for your architecture, to merge the attribute and issue any appropriate link warnings.
- * Test cases in 'ld/testsuite' for merging and link warnings.
- * 'binutils/readelf.c' to display your attribute.
- * GCC, if you want the compiler to mark the attribute automatically.



File: as.info, Node: Machine Dependencies, Next: Reporting Bugs, Prev: Object Attributes, Up: Top

9 Machine Dependent Features

The machine instruction sets are (almost by definition) different on each machine where 'as' runs. Floating point representations vary as well, and 'as' often supports a few additional directives or command-line options for compatibility with other assemblers on a

particular platform. Finally, some versions of 'as' support special pseudo-instructions for branch optimization.

This chapter discusses most of these differences, though it does not include details on any machine's instruction set. For details on that subject, see the hardware manufacturer's manual.

* Menu:

```
* AArch64-Dependent::      AArch64 Dependent Features
* Alpha-Dependent::       Alpha Dependent Features
* ARC-Dependent::         ARC Dependent Features
* ARM-Dependent::         ARM Dependent Features
* AVR-Dependent::         AVR Dependent Features
* Blackfin-Dependent::    Blackfin Dependent Features
* CR16-Dependent::        CR16 Dependent Features
* CRIS-Dependent::        CRIS Dependent Features
* D10V-Dependent::        D10V Dependent Features
* D30V-Dependent::        D30V Dependent Features
* Epiphany-Dependent::    EPIPHANY Dependent Features
* H8/300-Dependent::      Renesas H8/300 Dependent Features
* HPPA-Dependent::        HPPA Dependent Features
* ESA/390-Dependent::     IBM ESA/390 Dependent Features
* i386-Dependent::        Intel 80386 and AMD x86-64 Dependent Features
* i860-Dependent::        Intel 80860 Dependent Features
* i960-Dependent::        Intel 80960 Dependent Features
* IA-64-Dependent::       Intel IA-64 Dependent Features
* IP2K-Dependent::        IP2K Dependent Features
* LM32-Dependent::        LM32 Dependent Features
* M32C-Dependent::        M32C Dependent Features
* M32R-Dependent::        M32R Dependent Features
* M68K-Dependent::        M680x0 Dependent Features
* M68HC11-Dependent::     M68HC11 and 68HC12 Dependent Features
* Meta-Dependent ::       Meta Dependent Features
* MicroBlaze-Dependent::  MICROBLAZE Dependent Features
* MIPS-Dependent::        MIPS Dependent Features
* MMIX-Dependent::        MMIX Dependent Features
* MSP430-Dependent::      MSP430 Dependent Features
* NDS32-Dependent::       Andes NDS32 Dependent Features
* NiosII-Dependent::      Altera Nios II Dependent Features
* NS32K-Dependent::       NS32K Dependent Features
* PDP-11-Dependent::     PDP-11 Dependent Features
* PJ-Dependent::          picoJava Dependent Features
* PPC-Dependent::         PowerPC Dependent Features
* RL78-Dependent::        RL78 Dependent Features
* RISC-V-Dependent::     RISC-V Dependent Features
* RX-Dependent::          RX Dependent Features
* S/390-Dependent::      IBM S/390 Dependent Features
* SCORE-Dependent::      SCORE Dependent Features
* SH-Dependent::          Renesas / SuperH SH Dependent Features
* SH64-Dependent::       SuperH SH64 Dependent Features
* Sparc-Dependent::      SPARC Dependent Features
* TIC54X-Dependent::     TI TMS320C54x Dependent Features
* TIC6X-Dependent ::     TI TMS320C6x Dependent Features
* TILE-Gx-Dependent ::   Tilera TILE-Gx Dependent Features
* TILEPro-Dependent ::   Tilera TILEPro Dependent Features
* V850-Dependent::       V850 Dependent Features
* Vax-Dependent::        VAX Dependent Features
* Visium-Dependent::     Visium Dependent Features
```

```
* XGATE-Dependent::          XGATE Features
* XSTORMY16-Dependent::      XStormy16 Dependent Features
* Xtensa-Dependent::         Xtensa Dependent Features
* Z80-Dependent::            Z80 Dependent Features
* Z8000-Dependent::          Z8000 Dependent Features
```



File: as.info, Node: AArch64-Dependent, Next: Alpha-Dependent, Up: Machine Dependencies

9.1 AArch64 Dependent Features

=====

* Menu:

```
* AArch64 Options::          Options
* AArch64 Extensions::      Extensions
* AArch64 Syntax::          Syntax
* AArch64 Floating Point::  Floating Point
* AArch64 Directives::      AArch64 Machine Directives
* AArch64 Opcodes::         Opcodes
* AArch64 Mapping Symbols:: Mapping Symbols
```



File: as.info, Node: AArch64 Options, Next: AArch64 Extensions, Up: AArch64-Dependent

9.1.1 Options

'-EB'

This option specifies that the output generated by the assembler should be marked as being encoded for a big-endian processor.

'-EL'

This option specifies that the output generated by the assembler should be marked as being encoded for a little-endian processor.

'-mabi=ABI'

Specify which ABI the source code uses. The recognized arguments are: 'ilp32' and 'lp64', which decides the generated object file in ELF32 and ELF64 format respectively. The default is 'lp64'.

'-mcpu=PROCESSOR[+EXTENSION...]'

This option specifies the target processor. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target processor. The following processor names are recognized: 'cortex-a35', 'cortex-a53', 'cortex-a57', 'cortex-a72', 'cortex-a73', 'exynos-m1', 'falkor', 'qdf24xx', 'thunderx', 'vulcan', 'xgene1' and 'xgene2'. The special name 'all' may be used to allow the assembler to accept instructions valid for any supported processor, including all optional extensions.

In addition to the basic instruction set, the assembler can be told to accept, or restrict, various extension mnemonics that extend the processor. *Note AArch64 Extensions::.

If some implementations of a particular processor can have an

extension, then those extensions are automatically enabled. Consequently, you will not normally have to specify any additional extensions.

'-march=ARCHITECTURE[+EXTENSION...]'

This option specifies the target architecture. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target architecture. The following architecture names are recognized: 'armv8-a', 'armv8.1-a', 'armv8.2-a' and 'armv8.3-a'.

If both '-mcpu' and '-march' are specified, the assembler will use the setting for '-mcpu'. If neither are specified, the assembler will default to '-mcpu=all'.

The architecture option can be extended with the same instruction set extension options as the '-mcpu' option. Unlike '-mcpu', extensions are not always enabled by default, *Note AArch64 Extensions::.

'-mverbose-error'

This option enables verbose error messages for AArch64 gas. This option is enabled by default.

'-mno-verbose-error'

This option disables verbose error messages in AArch64 gas.



File: as.info, Node: AArch64 Extensions, Next: AArch64 Syntax, Prev: AArch64 Options, Up: AArch64-Dependent

9.1.2 Architecture Extensions

The table below lists the permitted architecture extensions that are supported by the assembler and the conditions under which they are automatically enabled.

Multiple extensions may be specified, separated by a '+'. Extension mnemonics may also be removed from those the assembler accepts. This is done by prepending 'no' to the option that adds the extension. Extensions that are removed must be listed after all extensions that have been added.

Enabling an extension that requires other extensions will automatically cause those extensions to be enabled. Similarly, disabling an extension that is required by other extensions will automatically cause those extensions to be disabled.

Extension	Minimum Architecture	Enabled by default	Description
'compnum'	ARMv8.2-A	ARMv8.3-A or later	Enable the complex number SIMD extensions. This implies 'fp16' and 'simd'.
'crc'	ARMv8-A	ARMv8.1-A or later	Enable CRC instructions.
'crypto'	ARMv8-A	No	Enable cryptographic extensions. This implies 'fp' and 'simd'.

'fp'	ARMv8-A	ARMv8-A or later	Enable floating-point extensions.
'fp16'	ARMv8.2-A	ARMv8.2-A or later	Enable ARMv8.2 16-bit floating-point support. This implies 'fp'.
'lor'	ARMv8-A	ARMv8.1-A or later	Enable Limited Ordering Regions extensions.
'lse'	ARMv8-A	ARMv8.1-A or later	Enable Large System extensions.
'pan'	ARMv8-A	ARMv8.1-A or later	Enable Privileged Access Never support.
'profile'	ARMv8.2-A	No	Enable statistical profiling extensions.
'ras'	ARMv8-A	ARMv8.2-A or later	Enable the Reliability, Availability and Serviceability extension.
'rdma'	ARMv8-A	ARMv8.1-A or later	Enable ARMv8.1 Advanced SIMD extensions. This implies 'simd'.
'simd'	ARMv8-A	ARMv8-A or later	Enable Advanced SIMD extensions. This implies 'fp'.
'sve'	ARMv8.2-A	No	Enable the Scalable Vector Extensions. This implies 'fp16', 'simd' and 'compnum'.



File: as.info, Node: AArch64 Syntax, Next: AArch64 Floating Point, Prev: AArch64 Extensions, Up: AArch64-Dependent

9.1.3 Syntax

* Menu:

- * AArch64-Chars:: Special Characters
- * AArch64-Regs:: Register Names
- * AArch64-Relocations:: Relocations



File: as.info, Node: AArch64-Chars, Next: AArch64-Regs, Up: AArch64 Syntax

9.1.3.1 Special Characters

.....

The presence of a '//' on a line indicates the start of a comment that extends to the end of the current line. If a '#' appears as the first character of a line, the whole line is treated as a comment.

The ';' character can be used instead of a newline to separate statements.

The '#' can be optionally used to indicate immediate operands.



File: as.info, Node: AArch64-Regs, Next: AArch64-Relocations, Prev: AArch64-Chars, Up: AArch64 Syntax

9.1.3.2 Register Names

.....

Please refer to the section '4.4 Register Names' of 'ARMv8 Instruction Set Overview', which is available at <<http://infocenter.arm.com>>.



File: as.info, Node: AArch64-Relocations, Prev: AArch64-Regs, Up: AArch64 Syntax

9.1.3.3 Relocations

.....

Relocations for 'MOVZ' and 'MOVK' instructions can be generated by prefixing the label with '#:abs_g2:' etc. For example to load the 48-bit absolute address of F00 into x0:

```
movz x0, #:abs_g2:foo      // bits 32-47, overflow check
movk x0, #:abs_g1_nc:foo  // bits 16-31, no overflow check
movk x0, #:abs_g0_nc:foo  // bits 0-15, no overflow check
```

Relocations for 'ADRP', and 'ADD', 'LDR' or 'STR' instructions can be generated by prefixing the label with ':pg_hi21:' and '#:lo12:' respectively.

For example to use 33-bit (+/-4GB) pc-relative addressing to load the address of F00 into x0:

```
adrp x0, :pg_hi21:foo
add x0, x0, #:lo12:foo
```

Or to load the value of F00 into x0:

```
adrp x0, :pg_hi21:foo
ldr x0, [x0, #:lo12:foo]
```

Note that ':pg_hi21:' is optional.

```
adrp x0, foo
```

is equivalent to

```
adrp x0, :pg_hi21:foo
```



File: as.info, Node: AArch64 Floating Point, Next: AArch64 Directives, Prev: AArch64 Syntax, Up: AArch64-Dependent

9.1.4 Floating Point

The AArch64 architecture uses IEEE floating-point numbers.



File: as.info, Node: AArch64 Directives, Next: AArch64 Opcodes, Prev: AArch64 Floating Point, Up: AArch64-Dependent

9.1.5 AArch64 Machine Directives

'**.arch** NAME'

Select the target architecture. Valid values for NAME are the same as for the '-march' commandline option.

Specifying '**.arch**' clears any previously selected architecture

extensions.

'`.arch_extension NAME`'

Add or remove an architecture extension to the target architecture. Valid values for NAME are the same as those accepted as architectural extensions by the '`-mcpu`' commandline option.

'`.arch_extension`' may be used multiple times to add or remove extensions incrementally to the architecture being compiled for.

'`.bss`'

This directive switches to the '`.bss`' section.

'`.cpu NAME`'

Set the target processor. Valid values for NAME are the same as those accepted by the '`-mcpu=`' command line option.

'`.dword EXPRESSIONS`'

The '`.dword`' directive produces 64 bit values.

'`.even`'

The '`.even`' directive aligns the output on the next even byte boundary.

'`.inst EXPRESSIONS`'

Inserts the expressions into the output as if they were instructions, rather than data.

'`.ltorg`'

This directive causes the current contents of the literal pool to be dumped into the current section (which is assumed to be the `.text` section) at the current location (aligned to a word boundary). GAS maintains a separate literal pool for each section and each sub-section. The '`.ltorg`' directive will only affect the literal pool of the current section and sub-section. At the end of assembly all remaining, un-empty literal pools will automatically be dumped.

Note - older versions of GAS would dump the current literal pool any time a section change occurred. This is no longer done, since it prevents accurate control of the placement of literal pools.

'`.pool`'

This is a synonym for `.ltorg`.

'`NAME .req REGISTER NAME`'

This creates an alias for REGISTER NAME called NAME. For example:

```
foo .req w0
```

'`.tlsdescadd`'

Emits a `TLSDESC_ADD` reloc on the next instruction.

'`.tlsdescall`'

Emits a `TLSDESC_CALL` reloc on the next instruction.

'`.tlsdescldr`'

Emits a `TLSDESC_LDR` reloc on the next instruction.

'`.unreq` ALIAS-NAME'

This undefines a register alias which was previously defined using the '`req`' directive. For example:

```
foo .req w0
.unreq foo
```

An error occurs if the name is undefined. Note - this pseudo op can be used to delete builtin in register name aliases (eg '`w0`'). This should only be done if it is really necessary.

'`.xword` EXPRESSIONS'

The '`.xword`' directive produces 64 bit values. This is the same as the '`.dword`' directive.



File: as.info, Node: AArch64 Opcodes, Next: AArch64 Mapping Symbols, Prev: AArch64 Directives, Up: AArch64-Dependent

9.1.6 Opcodes

GAS implements all the standard AArch64 opcodes. It also implements several pseudo opcodes, including several synthetic load instructions.

'LDR ='

```
ldr <register> , =<expression>
```

The constant expression will be placed into the nearest literal pool (if it not already there) and a PC-relative LDR instruction will be generated.

For more information on the AArch64 instruction set and assembly language notation, see 'ARMv8 Instruction Set Overview' available at <http://infocenter.arm.com>.



File: as.info, Node: AArch64 Mapping Symbols, Prev: AArch64 Opcodes, Up: AArch64-Dependent

9.1.7 Mapping Symbols

The AArch64 ELF specification requires that special symbols be inserted into object files to mark certain features:

'\$x'

At the start of a region of code containing AArch64 instructions.

'\$d'

At the start of a region of data.



File: as.info, Node: Alpha-Dependent, Next: ARC-Dependent, Prev: AArch64-Dependent, Up: Machine Dependencies

9.2 Alpha Dependent Features

=====

* Menu:

* Alpha Notes::	Notes
* Alpha Options::	Options
* Alpha Syntax::	Syntax
* Alpha Floating Point::	Floating Point
* Alpha Directives::	Alpha Machine Directives
* Alpha Opcodes::	Opcodes



File: as.info, Node: Alpha Notes, Next: Alpha Options, Up: Alpha-Dependent

9.2.1 Notes

The documentation here is primarily for the ELF object format. 'as' also supports the ECOFF and EVAX formats, but features specific to these formats are not yet documented.



File: as.info, Node: Alpha Options, Next: Alpha Syntax, Prev: Alpha Notes, Up: Alpha-Dependent

9.2.2 Options

'-mCPU'

This option specifies the target processor. If an attempt is made to assemble an instruction which will not execute on the target processor, the assembler may either expand the instruction as a macro or issue an error message. This option is equivalent to the '.arch' directive.

The following processor names are recognized: '21064', '21064a', '21066', '21068', '21164', '21164a', '21164pc', '21264', '21264a', '21264b', 'ev4', 'ev5', 'lca45', 'ev5', 'ev56', 'pca56', 'ev6', 'ev67', 'ev68'. The special name 'all' may be used to allow the assembler to accept instructions valid for any Alpha processor.

In order to support existing practice in OSF/1 with respect to '.arch', and existing practice within 'MILO' (the Linux ARC bootloader), the numbered processor names (e.g. 21064) enable the processor-specific PALcode instructions, while the "electro-vlasic" names (e.g. 'ev4') do not.

'-mdebug'

'-no-mdebug'

Enables or disables the generation of '.mdebug' encapsulation for stabs directives and procedure descriptors. The default is to automatically enable '.mdebug' when the first stabs directive is seen.

'-relax'

This option forces all relocations to be put into the object file, instead of saving space and resolving some relocations at assembly time. Note that this option does not propagate all symbol arithmetic into the object file, because not all symbol arithmetic can be represented. However, the option can still be useful in specific applications.

'-replace'

'-noreplace'

Enables or disables the optimization of procedure calls, both at assemblage and at link time. These options are only available for VMS targets and '-replace' is the default. See section 1.4.1 of the OpenVMS Linker Utility Manual.

'-g'

This option is used when the compiler generates debug information. When 'gcc' is using 'mips-tfile' to generate debug information for ECOFF, local labels must be passed through to the object file. Otherwise this option has no effect.

'-GSIZE'

A local common symbol larger than SIZE is placed in '.bss', while smaller symbols are placed in '.sbss'.

'-F'

'-32addr'

These options are ignored for backward compatibility.



File: as.info, Node: Alpha Syntax, Next: Alpha Floating Point, Prev: Alpha Options, Up: Alpha-Dependent

9.2.3 Syntax

The assembler syntax closely follow the Alpha Reference Manual; assembler directives and general syntax closely follow the OSF/1 and OpenVMS syntax, with a few differences for ELF.

* Menu:

* Alpha-Chars::	Special Characters
* Alpha-Regs::	Register Names
* Alpha-Relocs::	Relocations



File: as.info, Node: Alpha-Chars, Next: Alpha-Regs, Up: Alpha Syntax

9.2.3.1 Special Characters

.....

'#' is the line comment character. Note that if '#' is the first character on a line then it can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

';' can be used instead of a newline to separate statements.



File: as.info, Node: Alpha-Regs, Next: Alpha-Relocs, Prev: Alpha-Chars, Up: Alpha Syntax

9.2.3.2 Register Names

.....

The 32 integer registers are referred to as '\$N' or '\$rN'. In addition, registers 15, 28, 29, and 30 may be referred to by the symbols '\$fp', '\$at', '\$gp', and '\$sp' respectively.

The 32 floating-point registers are referred to as '\$fN'.



File: as.info, Node: Alpha-Relocs, Prev: Alpha-Regs, Up: Alpha Syntax

9.2.3.3 Relocations

.....

Some of these relocations are available for ECOFF, but mostly only for ELF. They are modeled after the relocation format introduced in Digital Unix 4.0, but there are additions.

The format is '!TAG' or '!TAG!NUMBER' where TAG is the name of the relocation. In some cases NUMBER is used to relate specific instructions.

The relocation is placed at the end of the instruction like so:

```
ldah $0,a($29)    !gprelhigh
lda  $0,a($0)     !gpellow
ldq  $1,b($29)    !literal!100
ldl  $2,0($1)     !lituse_base!100
```

'!literal'

'!literal!N'

Used with an 'ldq' instruction to load the address of a symbol from the GOT.

A sequence number N is optional, and if present is used to pair 'lituse' relocations with this 'literal' relocation. The 'lituse' relocations are used by the linker to optimize the code based on the final location of the symbol.

Note that these optimizations are dependent on the data flow of the program. Therefore, if `_any_ 'lituse'` is paired with a 'literal' relocation, then `_all_ uses` of the register set by the 'literal' instruction must also be marked with 'lituse' relocations. This is because the original 'literal' instruction may be deleted or transformed into another instruction.

Also note that there may be a one-to-many relationship between 'literal' and 'lituse', but not a many-to-one. That is, if there are two code paths that load up the same address and feed the value to a single use, then the use may not use a 'lituse' relocation.

'!lituse_base!N'

Used with any memory format instruction (e.g. 'ldl') to indicate that the literal is used for an address load. The offset field of the instruction must be zero. During relaxation, the code may be altered to use a gp-relative load.

'!lituse_jsr!N'

Used with a register branch format instruction (e.g. 'jsr') to indicate that the literal is used for a call. During relaxation, the code may be altered to use a direct branch (e.g. 'bsr').

`!lituse_jsrdirect!N'`

Similar to `!lituse_jsr'`, but also that this call cannot be vectored through a PLT entry. This is useful for functions with special calling conventions which do not allow the normal call-clobbered registers to be clobbered.

`!lituse_bytoff!N'`

Used with a byte mask instruction (e.g. `!extbl'`) to indicate that only the low 3 bits of the address are relevant. During relaxation, the code may be altered to use an immediate instead of a register shift.

`!lituse_addr!N'`

Used with any other instruction to indicate that the original address is in fact used, and the original `!ldq'` instruction may not be altered or deleted. This is useful in conjunction with `!lituse_jsr'` to test whether a weak symbol is defined.

```
ldq $27,foo($29)    !literal!1
beq $27,is_undef    !lituse_addr!1
jsr $26,($27),foo  !lituse_jsr!1
```

`!lituse_tlsugd!N'`

Used with a register branch format instruction to indicate that the literal is the call to `!__tls_get_addr'` used to compute the address of the thread-local storage variable whose descriptor was loaded with `!tlsugd!N'`.

`!lituse_tlsldm!N'`

Used with a register branch format instruction to indicate that the literal is the call to `!__tls_get_addr'` used to compute the address of the base of the thread-local storage block for the current module. The descriptor for the module must have been loaded with `!tlsldm!N'`.

`!gpdisp!N'`

Used with `!ldah'` and `!lda'` to load the GP from the current address, a-la the `!ldgp'` macro. The source register for the `!ldah'` instruction must contain the address of the `!ldah'` instruction. There must be exactly one `!lda'` instruction paired with the `!ldah'` instruction, though it may appear anywhere in the instruction stream. The immediate operands must be zero.

```
bsr $26,foo
ldah $29,0($26)    !gpdisp!1
lda $29,0($29)    !gpdisp!1
```

`!gprelhigh'`

Used with an `!ldah'` instruction to add the high 16 bits of a 32-bit displacement from the GP.

`!gprello'`

Used with any memory format instruction to add the low 16 bits of a 32-bit displacement from the GP.

`!gprel'`

Used with any memory format instruction to add a 16-bit displacement from the GP.

'!samegp'

Used with any branch format instruction to skip the GP load at the target address. The referenced symbol must have the same GP as the source object file, and it must be declared to either not use '\$27' or perform a standard GP load in the first two instructions via the '.prologue' directive.

'!tlsd'

'!tlsd!N'

Used with an 'lda' instruction to load the address of a TLS descriptor for a symbol in the GOT.

The sequence number N is optional, and if present it used to pair the descriptor load with both the 'literal' loading the address of the '__tls_get_addr' function and the 'lituse_tlsd' marking the call to that function.

For proper relaxation, both the 'tlsd', 'literal' and 'lituse' relocations must be in the same extended basic block. That is, the relocation with the lowest address must be executed first at runtime.

'!tlsldm'

'!tlsldm!N'

Used with an 'lda' instruction to load the address of a TLS descriptor for the current module in the GOT.

Similar in other respects to 'tlsd'.

'!gotdtprel'

Used with an 'ldq' instruction to load the offset of the TLS symbol within its module's thread-local storage block. Also known as the dynamic thread pointer offset or dtp-relative offset.

'!dtprelhi'

'!dtprello'

'!dtprel'

Like 'gprel' relocations except they compute dtp-relative offsets.

'!gottprel'

Used with an 'ldq' instruction to load the offset of the TLS symbol from the thread pointer. Also known as the tp-relative offset.

'!tprelhi'

'!tprello'

'!tprel'

Like 'gprel' relocations except they compute tp-relative offsets.



File: as.info, Node: Alpha Floating Point, Next: Alpha Directives, Prev: Alpha Syntax, Up: Alpha-Dependent

9.2.4 Floating Point

The Alpha family uses both IEEE and VAX floating-point numbers.



File: as.info, Node: Alpha Directives, Next: Alpha Opcodes, Prev: Alpha Floating Point, Up: Alpha-Dependent

9.2.5 Alpha Assembler Directives

'as' for the Alpha supports many additional directives for compatibility with the native assembler. This section describes them only briefly.

These are the additional directives in 'as' for the Alpha:

'`.arch CPU`'

Specifies the target processor. This is equivalent to the '`-mCPU`' command-line option. *Note Options: Alpha Options, for a list of values for CPU.

'`.ent FUNCTION[, N]`'

Mark the beginning of FUNCTION. An optional number may follow for compatibility with the OSF/1 assembler, but is ignored. When generating '`.mdebug`' information, this will create a procedure descriptor for the function. In ELF, it will mark the symbol as a function a-la the generic '`.type`' directive.

'`.end FUNCTION`'

Mark the end of FUNCTION. In ELF, it will set the size of the symbol a-la the generic '`.size`' directive.

'`.mask MASK, OFFSET`'

Indicate which of the integer registers are saved in the current function's stack frame. MASK is interpreted a bit mask in which bit N set indicates that register N is saved. The registers are saved in a block located OFFSET bytes from the "canonical frame address" (CFA) which is the value of the stack pointer on entry to the function. The registers are saved sequentially, except that the return address register (normally '`$26`') is saved first.

This and the other directives that describe the stack frame are currently only used when generating '`.mdebug`' information. They may in the future be used to generate DWARF2 '`.debug_frame`' unwind information for hand written assembly.

'`.fmask MASK, OFFSET`'

Indicate which of the floating-point registers are saved in the current stack frame. The MASK and OFFSET parameters are interpreted as with '`.mask`'.

'`.frame FRAMEREG, FRAMEOFFSET, RETREG[, ARGOFFSET]`'

Describes the shape of the stack frame. The frame pointer in use is FRAMEREG; normally this is either '`$fp`' or '`$sp`'. The frame pointer is FRAMEOFFSET bytes below the CFA. The return address is initially located in RETREG until it is saved as indicated in '`.mask`'. For compatibility with OSF/1 an optional ARGOFFSET parameter is accepted and ignored. It is believed to indicate the offset from the CFA to the saved argument registers.

'`.prologue N`'

Indicate that the stack frame is set up and all registers have been spilled. The argument N indicates whether and how the function uses the incoming "procedure vector" (the address of the called

function) in '\$27'. 0 indicates that '\$27' is not used; 1 indicates that the first two instructions of the function use '\$27' to perform a load of the GP register; 2 indicates that '\$27' is used in some non-standard way and so the linker cannot elide the load of the procedure vector during relaxation.

'`.usepv FUNCTION, WHICH`'

Used to indicate the use of the '\$27' register, similar to '`.prologue`', but without the other semantics of needing to be inside an open '`.ent`'/'`.end`' block.

The WHICH argument should be either 'no', indicating that '\$27' is not used, or 'std', indicating that the first two instructions of the function perform a GP load.

One might use this directive instead of '`.prologue`' if you are also using dwarf2 CFI directives.

'`.gprel32 EXPRESSION`'

Computes the difference between the address in EXPRESSION and the GP for the current object file, and stores it in 4 bytes. In addition to being smaller than a full 8 byte address, this also does not require a dynamic relocation when used in a shared library.

'`.t_floating EXPRESSION`'

Stores EXPRESSION as an IEEE double precision value.

'`.s_floating EXPRESSION`'

Stores EXPRESSION as an IEEE single precision value.

'`.f_floating EXPRESSION`'

Stores EXPRESSION as a VAX F format value.

'`.g_floating EXPRESSION`'

Stores EXPRESSION as a VAX G format value.

'`.d_floating EXPRESSION`'

Stores EXPRESSION as a VAX D format value.

'`.set FEATURE`'

Enables or disables various assembler features. Using the positive name of the feature enables while using 'noFEATURE' disables.

'at'

Indicates that macro expansions may clobber the "assembler temporary" ('\$at' or '\$28') register. Some macros may not be expanded without this and will generate an error message if 'noat' is in effect. When 'at' is in effect, a warning will be generated if '\$at' is used by the programmer.

'macro'

Enables the expansion of macro instructions. Note that variants of real instructions, such as 'br label' vs 'br \$31,label' are considered alternate forms and not macros.

'move'

'reorder'

'volatile'

These control whether and how the assembler may re-order instructions. Accepted for compatibility with the OSF/1 assembler, but 'as' does not do instruction scheduling, so these features are ignored.

The following directives are recognized for compatibility with the OSF/1 assembler but are ignored.

```
.proc          .aproc
.reguse       .livereg
.option       .aent
.ugen        .eflag
.alias       .noalias
```



File: as.info, Node: Alpha Opcodes, Prev: Alpha Directives, Up: Alpha-Dependent

9.2.6 Opcodes

For detailed information on the Alpha machine instruction set, see the Alpha Architecture Handbook (<ftp://ftp.digital.com/pub/Digital/info/semiconductor/literature/alphaahb.pdf>).



File: as.info, Node: ARC-Dependent, Next: ARM-Dependent, Prev: Alpha-Dependent, Up: Machine Dependencies

9.3 ARC Dependent Features

=====

* Menu:

```
* ARC Options::          Options
* ARC Syntax::          Syntax
* ARC Directives::      ARC Machine Directives
* ARC Modifiers::       ARC Assembler Modifiers
* ARC Symbols::         ARC Pre-defined Symbols
* ARC Opcodes::         Opcodes
```



File: as.info, Node: ARC Options, Next: ARC Syntax, Up: ARC-Dependent

9.3.1 Options

The following options control the type of CPU for which code is assembled, and generic constraints on the code generated:

'-mcpu=CPU'

Set architecture type and register usage for CPU. There are also shortcut alias options available for backward compatibility and convenience. Supported values for CPU are

'arc600'

Assemble for ARC 600. Aliases: '-mA6', '-mARC600'.

'arc600_norm'

Assemble for ARC 600 with norm instructions.

'arc600_mul64'
Assemble for ARC 600 with mul64 instructions.

'arc600_mul32x16'
Assemble for ARC 600 with mul32x16 instructions.

'arc601'
Assemble for ARC 601. Alias: '-mARC601'.

'arc601_norm'
Assemble for ARC 601 with norm instructions.

'arc601_mul64'
Assemble for ARC 601 with mul64 instructions.

'arc601_mul32x16'
Assemble for ARC 601 with mul32x16 instructions.

'arc700'
Assemble for ARC 700. Aliases: '-mA7', '-mARC700'.

'arcem'
Assemble for ARC EM. Aliases: '-mEM'

'em'
Assemble for ARC EM, identical as arcem variant.

'em4'
Assemble for ARC EM with code-density instructions.

'em4_dmips'
Assemble for ARC EM with code-density instructions.

'em4_fpus'
Assemble for ARC EM with code-density instructions.

'em4_fpuda'
Assemble for ARC EM with code-density, and double-precision assist instructions.

'quarkse_em'
Assemble for QuarkSE-EM cpu.

'archs'
Assemble for ARC HS. Aliases: '-mHS', '-mav2hs'.

'hs'
Assemble for ARC HS.

'hs34'
Assemble for ARC HS34.

'hs38'
Assemble for ARC HS38.

'hs38_linux'
Assemble for ARC HS38 with floating point support on.

'nps400'

Assemble for ARC 700 with NPS-400 extended instructions.

Note: the '.cpu' directive (*note ARC Directives::) can to be used to select a core variant from within assembly code.

'-EB'

This option specifies that the output generated by the assembler should be marked as being encoded for a big-endian processor.

'-EL'

This option specifies that the output generated by the assembler should be marked as being encoded for a little-endian processor - this is the default.

'-mcode-density'

This option turns on Code Density instructions. Only valid for ARC EM processors.

'-mrelax'

Enable support for assembly-time relaxation. The assembler will replace a longer version of an instruction with a shorter one, whenever it is possible.

'-mnps400'

Enable support for NPS-400 extended instructions.

'-mspfp'

Enable support for single-precision floating point instructions.

'-mdpfp'

Enable support for double-precision floating point instructions.

'-mfpuda'

Enable support for double-precision assist floating point instructions. Only valid for ARC EM processors.



File: as.info, Node: ARC Syntax, Next: ARC Directives, Prev: ARC Options, Up: ARC-Dependent

9.3.2 Syntax

* Menu:

* ARC-Chars:: Special Characters

* ARC-Regs:: Register Names



File: as.info, Node: ARC-Chars, Next: ARC-Regs, Up: ARC Syntax

9.3.2.1 Special Characters

.....

'%'

A register name can optionally be prefixed by a '%' character. So register '%r0' is equivalent to 'r0' in the assembly code.

'#'

The presence of a '#' character within a line (but not at the start of a line) indicates the start of a comment that extends to the end of the current line.

Note: if a line starts with a '#' character then it can also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

'@'

Prefixing an operand with an '@' specifies that the operand is a symbol and not a register. This is how the assembler disambiguates the use of an ARC register name as a symbol. So the instruction

```
mov r0, @r0
```

moves the address of symbol 'r0' into register 'r0'.

'`'

The '`' (backtick) character is used to separate statements on a single line.

'_'

Used as a separator to obtain a sequence of commands from a C preprocessor macro.



File: as.info, Node: ARC-Regs, Prev: ARC-Chars, Up: ARC Syntax

9.3.2.2 Register Names

.....

The ARC assembler uses the following register names for its core registers:

'r0-r31'

The core general registers. Registers 'r26' through 'r31' have special functions, and are usually referred to by those synonyms.

'gp'

The global pointer and a synonym for 'r26'.

'fp'

The frame pointer and a synonym for 'r27'.

'sp'

The stack pointer and a synonym for 'r28'.

'ilink1'

For ARC 600 and ARC 700, the level 1 interrupt link register and a synonym for 'r29'. Not supported for ARCV2.

'ilink'

For ARCV2, the interrupt link register and a synonym for 'r29'. Not supported for ARC 600 and ARC 700.

'ilink2'

For ARC 600 and ARC 700, the level 2 interrupt link register and a synonym for 'r30'. Not supported for ARC v2.

'blink'

The link register and a synonym for 'r31'.

'r32-r59'

The extension core registers.

'lp_count'

The loop count register.

'pcl'

The word aligned program counter.

In addition the ARC processor has a large number of `_auxiliary registers_`. The precise set depends on the extensions being supported, but the following baseline set are always defined:

'identity'

Processor Identification register. Auxiliary register address 0x4.

'pc'

Program Counter. Auxiliary register address 0x6.

'status32'

Status register. Auxiliary register address 0x0a.

'bta'

Branch Target Address. Auxiliary register address 0x412.

'ecr'

Exception Cause Register. Auxiliary register address 0x403.

'int_vector_base'

Interrupt Vector Base address. Auxiliary register address 0x25.

'status32_p0'

Stored STATUS32 register on entry to level P0 interrupts.
Auxiliary register address 0xb.

'aux_user_sp'

Saved User Stack Pointer. Auxiliary register address 0xd.

'eret'

Exception Return Address. Auxiliary register address 0x400.

'erbta'

BTA saved on exception entry. Auxiliary register address 0x401.

'erstatus'

STATUS32 saved on exception. Auxiliary register address 0x402.

'bcr_ver'

Build Configuration Registers Version. Auxiliary register address
0x60.

'bta_link_build'

Build configuration for: BTA Registers. Auxiliary register address
0x63.

'vecbase_ac_build'

Build configuration for: Interrupts. Auxiliary register address

0x68.

'rf_build'

Build configuration for: Core Registers. Auxiliary register address 0x6e.

'dccm_build'

DCCM RAM Configuration Register. Auxiliary register address 0xc1.

Additional auxiliary register names are defined according to the processor architecture version and extensions selected by the options.



File: as.info, Node: ARC Directives, Next: ARC Modifiers, Prev: ARC Syntax, Up: ARC-Dependent

9.3.3 ARC Machine Directives

The ARC version of 'as' supports the following additional machine directives:

'**.lcomm** SYMBOL, LENGTH[, ALIGNMENT]'

Reserve LENGTH (an absolute expression) bytes for a local common denoted by SYMBOL. The section and value of SYMBOL are those of the new local common. The addresses are allocated in the bss section, so that at run-time the bytes start off zeroed. Since SYMBOL is not declared global, it is normally not visible to 'ld'. The optional third parameter, ALIGNMENT, specifies the desired alignment of the symbol in the bss section, specified as a byte boundary (for example, an alignment of 16 means that the least significant 4 bits of the address should be zero). The alignment must be an absolute expression, and it must be a power of two. If no alignment is specified, as will set the alignment to the largest power of two less than or equal to the size of the symbol, up to a maximum of 16.

'**.lcommon** SYMBOL, LENGTH[, ALIGNMENT]'

The same as 'lcomm' directive.

'**.cpu** CPU'

The '.cpu' directive must be followed by the desired core version. Permitted values for CPU are:

'ARC600'

Assemble for the ARC600 instruction set.

'arc600_norm'

Assemble for ARC 600 with norm instructions.

'arc600_mul64'

Assemble for ARC 600 with mul64 instructions.

'arc600_mul32x16'

Assemble for ARC 600 with mul32x16 instructions.

'arc601'

Assemble for ARC 601 instruction set.

'arc601_norm'

Assemble for ARC 601 with norm instructions.

'arc601_mul64'

Assemble for ARC 601 with mul64 instructions.

'arc601_mul32x16'

Assemble for ARC 601 with mul32x16 instructions.

'ARC700'

Assemble for the ARC700 instruction set.

'NPS400'

Assemble for the NPS400 instruction set.

'EM'

Assemble for the ARC EM instruction set.

'arcem'

Assemble for ARC EM instruction set

'em4'

Assemble for ARC EM with code-density instructions.

'em4_dmips'

Assemble for ARC EM with code-density instructions.

'em4_fpus'

Assemble for ARC EM with code-density instructions.

'em4_fpuda'

Assemble for ARC EM with code-density, and double-precision assist instructions.

'quarkse_em'

Assemble for QuarkSE-EM instruction set.

'HS'

Assemble for the ARC HS instruction set.

'archs'

Assemble for ARC HS instruction set.

'hs'

Assemble for ARC HS instruction set.

'hs34'

Assemble for ARC HS34 instruction set.

'hs38'

Assemble for ARC HS38 instruction set.

'hs38_linux'

Assemble for ARC HS38 with floating point support on.

Note: the '.cpu' directive overrides the command line option '-mcpu=CPU'; a warning is emitted when the version is not consistent between the two.

'.extAuxRegister NAME, ADDR, MODE'

Auxiliary registers can be defined in the assembler source code by using this directive. The first parameter, NAME, is the name of the new auxiliary register. The second parameter, ADDR, is address of the auxiliary register. The third parameter, MODE, specifies whether the register is readable and/or writable and is one of:

```
'r'
    Read only;

'w'
    Write only;

'r|w'
    Read and write.
```

For example:

```
.extAuxRegister mulhi, 0x12, w
specifies a write only extension auxiliary register, MULHI at
address 0x12.
```

'`.extCondCode SUFFIX, VAL`'

ARC supports extensible condition codes. This directive defines a new condition code, to be known by the suffix, SUFFIX and will depend on the value, VAL in the condition code.

For example:

```
.extCondCode is_busy,0x14
add.is_busy r1,r2,r3
will only execute the 'add' instruction if the condition code value
is 0x14.
```

'`.extCoreRegister NAME, REGNUM, MODE, SHORTCUT`'

Specifies an extension core register named NAME as a synonym for the register numbered REGNUM. The register number must be between 32 and 59. The third argument, MODE, indicates whether the register is readable and/or writable and is one of:

```
'r'
    Read only;

'w'
    Write only;

'r|w'
    Read and write.
```

The final parameter, SHORTCUT indicates whether the register has a short cut in the pipeline. The valid values are:

```
'can_shortcut'
    The register has a short cut in the pipeline;

'cannot_shortcut'
    The register does not have a short cut in the pipeline.
```

For example:

```
.extCoreRegister mlo, 57, r , can_shortcut
defines a read only extension core register, 'mlo', which is
register 57, and can short cut the pipeline.
```

'`.extInstruction NAME, OPCODE, SUBOPCODE, SUFFIXCLASS, SYNTAXCLASS`'

ARC allows the user to specify extension instructions. These extension instructions are not macros; the assembler creates encodings for use of these instructions according to the specification by the user.

The first argument, NAME, gives the name of the instruction.

The second argument, OPCODE, is the opcode to be used (bits 31:27 in the encoding).

The third argument, SUBOPCODE, is the sub-opcode to be used, but the correct value also depends on the fifth argument, SYNTAXCLASS

The fourth argument, SUFFIXCLASS, determines the kinds of suffixes to be allowed. Valid values are:

'SUFFIX_NONE'

No suffixes are permitted;

'SUFFIX_COND'

Conditional suffixes are permitted;

'SUFFIX_FLAG'

Flag setting suffixes are permitted.

'SUFFIX_COND|SUFFIX_FLAG'

Both conditional and flag setting suffices are permitted.

The fifth and final argument, SYNTAXCLASS, determines the syntax class for the instruction. It can have the following values:

'SYNTAX_2OP'

Two Operand Instruction;

'SYNTAX_3OP'

Three Operand Instruction.

'SYNTAX_1OP'

One Operand Instruction.

'SYNTAX_NOP'

No Operand Instruction.

The syntax class may be followed by '|' and one of the following modifiers.

'OP1_MUST_BE_IMM'

Modifies syntax class 'SYNTAX_3OP', specifying that the first operand of a three-operand instruction must be an immediate (i.e., the result is discarded). This is usually used to set the flags using specific instructions and not retain results.

'OP1_IMM_IMPLIED'

Modifies syntax class 'SYNTAX_2OP', specifying that there is an implied immediate destination operand which does not appear in the syntax.

For example, if the source code contains an instruction like:

```
inst r1,r2
```

the first argument is an implied immediate (that is, the result is discarded). This is the same as though the source

code were: `inst 0,r1,r2.`

For example, defining a 64-bit multiplier with immediate operands:

```
.extInstruction mp64, 0x07, 0x2d, SUFFIX_COND|SUFFIX_FLAG,
    SYNTAX_3OP|OP1_MUST_BE_IMM
```

which specifies an extension instruction named 'mp64' with 3 operands. It sets the flags and can be used with a condition code, for which the first operand is an immediate, i.e. equivalent to discarding the result of the operation.

A two operands instruction variant would be:

```
.extInstruction mul64, 0x07, 0x2d, SUFFIX_COND,
    SYNTAX_2OP|OP1_IMM_IMPLIED
```

which describes a two operand instruction with an implicit first immediate operand. The result of this operation would be discarded.



File: as.info, Node: ARC Modifiers, Next: ARC Symbols, Prev: ARC Directives, Up: ARC-Dependent

9.3.4 ARC Assembler Modifiers

The following additional assembler modifiers have been added for position-independent code. These modifiers are available only with the ARC 700 and above processors and generate relocation entries, which are interpreted by the linker as follows:

'@pcl(SYMBOL)'

Relative distance of SYMBOL's from the current program counter location.

'@gotpc(SYMBOL)'

Relative distance of SYMBOL's Global Offset Table entry from the current program counter location.

'@gotoff(SYMBOL)'

Distance of SYMBOL from the base of the Global Offset Table.

'@plt(SYMBOL)'

Distance of SYMBOL's Procedure Linkage Table entry from the current program counter. This is valid only with branch and link instructions and PC-relative calls.

'@sda(SYMBOL)'

Relative distance of SYMBOL from the base of the Small Data Pointer.



File: as.info, Node: ARC Symbols, Next: ARC Opcodes, Prev: ARC Modifiers, Up: ARC-Dependent

9.3.5 ARC Pre-defined Symbols

The following assembler symbols will prove useful when developing position-independent code. These symbols are available only with the ARC 700 and above processors.

'__GLOBAL_OFFSET_TABLE__'

Symbol referring to the base of the Global Offset Table.

'__DYNAMIC__'

An alias for the Global Offset Table 'Base__GLOBAL_OFFSET_TABLE__'.
It can be used only with '@gotpc' modifiers.

US

File: as.info, Node: ARC Opcodes, Prev: ARC Symbols, Up: ARC-Dependent

9.3.6 Opcodes

For information on the ARC instruction set, see 'ARC Programmers Reference Manual', available where you download the processor IP library.

US

File: as.info, Node: ARM-Dependent, Next: AVR-Dependent, Prev: ARC-Dependent, Up: Machine Dependencies

9.4 ARM Dependent Features

=====

* Menu:

* ARM Options::	Options
* ARM Syntax::	Syntax
* ARM Floating Point::	Floating Point
* ARM Directives::	ARM Machine Directives
* ARM Opcodes::	Opcodes
* ARM Mapping Symbols::	Mapping Symbols
* ARM Unwinding Tutorial::	Unwinding

US

File: as.info, Node: ARM Options, Next: ARM Syntax, Up: ARM-Dependent

9.4.1 Options

'-mcpu=PROCESSOR[+EXTENSION...]'

This option specifies the target processor. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target processor. The following processor names are recognized: 'arm1', 'arm2', 'arm250', 'arm3', 'arm6', 'arm60', 'arm600', 'arm610', 'arm620', 'arm7', 'arm7m', 'arm7d', 'arm7dm', 'arm7di', 'arm7dmi', 'arm70', 'arm700', 'arm700i', 'arm710', 'arm710t', 'arm720', 'arm720t', 'arm740t', 'arm710c', 'arm7100', 'arm7500', 'arm7500fe', 'arm7t', 'arm7tdmi', 'arm7tdmi-s', 'arm8', 'arm810', 'strongarm', 'strongarm1', 'strongarm110', 'strongarm1100', 'strongarm1110', 'arm9', 'arm920', 'arm920t', 'arm922t', 'arm940t', 'arm9tdmi', 'fa526' (Faraday FA526 processor), 'fa626' (Faraday FA626 processor), 'arm9e', 'arm926e', 'arm926ej-s', 'arm946e-r0', 'arm946e', 'arm946e-s', 'arm966e-r0', 'arm966e', 'arm966e-s', 'arm968e-s', 'arm10t', 'arm10tdmi', 'arm10e', 'arm1020', 'arm1020t', 'arm1020e', 'arm1022e', 'arm1026ej-s', 'fa606te' (Faraday FA606TE processor), 'fa616te' (Faraday FA616TE processor), 'fa626te' (Faraday FA626TE processor),

'fmp626' (Faraday FMP626 processor), 'fa726te' (Faraday FA726TE processor), 'arm1136j-s', 'arm1136jf-s', 'arm1156t2-s', 'arm1156t2f-s', 'arm1176jz-s', 'arm1176jzf-s', 'mpcore', 'mpcorenovfp', 'cortex-a5', 'cortex-a7', 'cortex-a8', 'cortex-a9', 'cortex-a15', 'cortex-a17', 'cortex-a32', 'cortex-a35', 'cortex-a53', 'cortex-a57', 'cortex-a72', 'cortex-a73', 'cortex-r4', 'cortex-r4f', 'cortex-r5', 'cortex-r7', 'cortex-r8', 'cortex-m33', 'cortex-m23', 'cortex-m7', 'cortex-m4', 'cortex-m3', 'cortex-m1', 'cortex-m0', 'cortex-m0plus', 'exynos-m1', 'marvell-pj4', 'marvell-whitney', 'falkor', 'qdf24xx', 'xgene1', 'xgene2', 'ep9312' (ARM920 with Cirrus Maverick coprocessor), 'i80200' (Intel XScale processor) 'iwmmxt' (Intel(r) XScale processor with Wireless MMX(tm) technology coprocessor) and 'xscale'. The special name 'all' may be used to allow the assembler to accept instructions valid for any ARM processor.

In addition to the basic instruction set, the assembler can be told to accept various extension mnemonics that extend the processor using the co-processor instruction space. For example, '-mcpu=arm920+maverick' is equivalent to specifying '-mcpu=ep9312'.

Multiple extensions may be specified, separated by a '+'. The extensions should be specified in ascending alphabetical order.

Some extensions may be restricted to particular architectures; this is documented in the list of extensions below.

Extension mnemonics may also be removed from those the assembler accepts. This is done by prepending 'no' to the option that adds the extension. Extensions that are removed should be listed after all extensions which have been added, again in ascending alphabetical order. For example, '-mcpu=ep9312+nomaverick' is equivalent to specifying '-mcpu=arm920'.

The following extensions are currently supported: 'crc' (Cryptography Extensions for v8-A architecture, implies 'fp+simd'), 'fp' (Floating Point Extensions for v8-A architecture), 'idiv' (Integer Divide Extensions for v7-A and v7-R architectures), 'iwmmxt', 'iwmmxt2', 'xscale', 'maverick', 'mp' (Multiprocessing Extensions for v7-A and v7-R architectures), 'os' (Operating System for v6M architecture), 'sec' (Security Extensions for v6K and v7-A architectures), 'simd' (Advanced SIMD Extensions for v8-A architecture, implies 'fp'), 'virt' (Virtualization Extensions for v7-A architecture, implies 'idiv'), 'pan' (Privileged Access Never Extensions for v8-A architecture), 'ras' (Reliability, Availability and Serviceability extensions for v8-A architecture), 'rdma' (ARMv8.1 Advanced SIMD extensions for v8-A architecture, implies 'simd') and 'xscale'.

'-march=ARCHITECTURE[+EXTENSION...]'

This option specifies the target architecture. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target architecture. The following architecture names are recognized: 'armv1', 'armv2', 'armv2a', 'armv2s', 'armv3', 'armv3m', 'armv4', 'armv4xm', 'armv4t', 'armv4txm', 'armv5', 'armv5t', 'armv5txm', 'armv5te', 'armv5texp', 'armv6', 'armv6j', 'armv6k', 'armv6z', 'armv6kz', 'armv6-m', 'armv6s-m', 'armv7', 'armv7-a', 'armv7ve', 'armv7-r', 'armv7-m', 'armv7e-m', 'armv8-a', 'armv8.1-a', 'armv8.2-a',

'armv8.3-a', 'iwmmxt', 'iwmmxt2' and 'xscale'. If both '-mcpu' and '-march' are specified, the assembler will use the setting for '-mcpu'.

The architecture option can be extended with the same instruction set extension options as the '-mcpu' option.

'-mfpu=FLOATING-POINT-FORMAT'

This option specifies the floating point format to assemble for. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target floating point unit. The following format options are recognized: 'softfpa', 'fpe', 'fpe2', 'fpe3', 'fpa', 'fpa10', 'fpa11', 'arm7500fe', 'softvfp', 'softvfp+vfp', 'vfp', 'vfp10', 'vfp10-r0', 'vfp9', 'vfp9d', 'vfpv2', 'vfpv3', 'vfpv3-fp16', 'vfpv3-d16', 'vfpv3-d16-fp16', 'vfpv3xd', 'vfpv3xd-d16', 'vfpv4', 'vfpv4-d16', 'fpv4-sp-d16', 'fpv5-sp-d16', 'fpv5-d16', 'fp-armv8', 'arm1020t', 'arm1020e', 'arm1136jf-s', 'maverick', 'neon', 'neon-vfpv4', 'neon-fp-armv8', 'crypto-neon-fp-armv8', 'neon-fp-armv8.1' and 'crypto-neon-fp-armv8.1'.

In addition to determining which instructions are assembled, this option also affects the way in which the '.double' assembler directive behaves when assembling little-endian code.

The default is dependent on the processor selected. For Architecture 5 or later, the default is to assembler for VFP instructions; for earlier architectures the default is to assemble for FPA instructions.

'-mthumb'

This option specifies that the assembler should start assembling Thumb instructions; that is, it should behave as though the file starts with a '.code 16' directive.

'-mthumb-interwork'

This option specifies that the output generated by the assembler should be marked as supporting interworking.

'-mimplicit-it=never'

'-mimplicit-it=always'

'-mimplicit-it=arm'

'-mimplicit-it=thumb'

The '-mimplicit-it' option controls the behavior of the assembler when conditional instructions are not enclosed in IT blocks. There are four possible behaviors. If 'never' is specified, such constructs cause a warning in ARM code and an error in Thumb-2 code. If 'always' is specified, such constructs are accepted in both ARM and Thumb-2 code, where the IT instruction is added implicitly. If 'arm' is specified, such constructs are accepted in ARM code and cause an error in Thumb-2 code. If 'thumb' is specified, such constructs cause a warning in ARM code and are accepted in Thumb-2 code. If you omit this option, the behavior is equivalent to '-mimplicit-it=arm'.

'-mapcs-26'

'-mapcs-32'

These options specify that the output generated by the assembler

should be marked as supporting the indicated version of the Arm Procedure Calling Standard.

'-matpcs'

This option specifies that the output generated by the assembler should be marked as supporting the Arm/Thumb Procedure Calling Standard. If enabled this option will cause the assembler to create an empty debugging section in the object file called `.arm.atpcs`. Debuggers can use this to determine the ABI being used by.

'-mapcs-float'

This indicates the floating point variant of the APCS should be used. In this variant floating point arguments are passed in FP registers rather than integer registers.

'-mapcs-reentrant'

This indicates that the reentrant variant of the APCS should be used. This variant supports position independent code.

'-mfloat-abi=ABI'

This option specifies that the output generated by the assembler should be marked as using specified floating point ABI. The following values are recognized: 'soft', 'softfp' and 'hard'.

'-meabi=VER'

This option specifies which EABI version the produced object files should conform to. The following values are recognized: 'gnu', '4' and '5'.

'-EB'

This option specifies that the output generated by the assembler should be marked as being encoded for a big-endian processor.

Note: If a program is being built for a system with big-endian data and little-endian instructions then it should be assembled with the '-EB' option, (all of it, code and data) and then linked with the '--be8' option. This will reverse the endianness of the instructions back to little-endian, but leave the data as big-endian.

'-EL'

This option specifies that the output generated by the assembler should be marked as being encoded for a little-endian processor.

'-k'

This option specifies that the output of the assembler should be marked as position-independent code (PIC).

'--fix-v4bx'

Allow 'BX' instructions in ARMv4 code. This is intended for use with the linker option of the same name.

'-mwarn-deprecated'

'-mno-warn-deprecated'

Enable or disable warnings about using deprecated options or features. The default is to warn.

'-mccs'

Turns on CodeComposer Studio assembly syntax compatibility mode.

'-mwarn-syms'

'-mno-warn-syms'

Enable or disable warnings about symbols that match the names of ARM instructions. The default is to warn.



File: as.info, Node: ARM Syntax, Next: ARM Floating Point, Prev: ARM Options, Up: ARM-Dependent

9.4.2 Syntax

* Menu:

- * ARM-Instruction-Set:: Instruction Set
- * ARM-Chars:: Special Characters
- * ARM-Regs:: Register Names
- * ARM-Relocations:: Relocations
- * ARM-Neon-Alignment:: NEON Alignment Specifiers



File: as.info, Node: ARM-Instruction-Set, Next: ARM-Chars, Up: ARM Syntax

9.4.2.1 Instruction Set Syntax

.....

Two slightly different syntaxes are support for ARM and THUMB instructions. The default, 'divided', uses the old style where ARM and THUMB instructions had their own, separate syntaxes. The new, 'unified' syntax, which can be selected via the '.syntax' directive, and has the following main features:

- * Immediate operands do not require a '#' prefix.
- * The 'IT' instruction may appear, and if it does it is validated against subsequent conditional affixes. In ARM mode it does not generate machine code, in THUMB mode it does.
- * For ARM instructions the conditional affixes always appear at the end of the instruction. For THUMB instructions conditional affixes can be used, but only inside the scope of an 'IT' instruction.
- * All of the instructions new to the V6T2 architecture (and later) are available. (Only a few such instructions can be written in the 'divided' syntax).
- * The '.N' and '.W' suffixes are recognized and honored.
- * All instructions set the flags if and only if they have an 's' affix.



File: as.info, Node: ARM-Chars, Next: ARM-Regs, Prev: ARM-Instruction-Set, Up: ARM Syntax

9.4.2.2 Special Characters

.....

The presence of a '@' anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

The ';' character can be used instead of a newline to separate statements.

Either '#' or '\$' can be used to indicate immediate operands.

TODO Explain about /data modifier on symbols.



File: as.info, Node: ARM-Regs, Next: ARM-Relocations, Prev: ARM-Chars, Up: ARM Syntax

9.4.2.3 Register Names

.....

TODO Explain about ARM register naming, and the predefined names.



File: as.info, Node: ARM-Relocations, Next: ARM-Neon-Alignment, Prev: ARM-Regs, Up: ARM Syntax

9.4.2.4 ARM relocation generation

.....

Specific data relocations can be generated by putting the relocation name in parentheses after the symbol name. For example:

```
.word foo(TARGET1)
```

This will generate an 'R_ARM_TARGET1' relocation against the symbol FOO. The following relocations are supported: 'GOT', 'GOTOFF', 'TARGET1', 'TARGET2', 'SBREL', 'TLSGD', 'TSLDM', 'TSLDO', 'TSLDESC', 'TSCALL', 'GOTTPOFF', 'GOT_PREL' and 'TPOFF'.

For compatibility with older toolchains the assembler also accepts '(PLT)' after branch targets. On legacy targets this will generate the deprecated 'R_ARM_PLT32' relocation. On EABI targets it will encode either the 'R_ARM_CALL' or 'R_ARM_JUMP24' relocation, as appropriate.

Relocations for 'MOVW' and 'MOVT' instructions can be generated by prefixing the value with '#:lower16:' and '#:upper16' respectively. For example to load the 32-bit address of foo into r0:

```
MOVW r0, #:lower16:foo
MOVT r0, #:upper16:foo
```

Relocations 'R_ARM_THM_ALU_ABS_G0_NC', 'R_ARM_THM_ALU_ABS_G1_NC', 'R_ARM_THM_ALU_ABS_G2_NC' and 'R_ARM_THM_ALU_ABS_G3_NC' can be generated by prefixing the value with '#:lower0_7:#', '#:lower8_15:#', '#:upper0_7:#' and '#:upper8_15:#' respectively. For example to load the 32-bit address of foo into r0:

```

MOVSW r0, #:upper8_15:#foo
LSLS r0, r0, #8
ADDS r0, #:upper0_7:#foo
LSLS r0, r0, #8
ADDS r0, #:lower8_15:#foo
LSLS r0, r0, #8
ADDS r0, #:lower0_7:#foo

```



File: as.info, Node: ARM-Neon-Alignment, Prev: ARM-Relocations, Up: ARM Syntax

9.4.2.5 NEON Alignment Specifiers

.....

Some NEON load/store instructions allow an optional address alignment qualifier. The ARM documentation specifies that this is indicated by '@ALIGN'. However GAS already interprets the '@' character as a "line comment" start, so ': ALIGN' is used instead. For example:

```
vld1.8 {q0}, [r0, :128]
```



File: as.info, Node: ARM Floating Point, Next: ARM Directives, Prev: ARM Syntax, Up: ARM-Dependent

9.4.3 Floating Point

The ARM family uses IEEE floating-point numbers.



File: as.info, Node: ARM Directives, Next: ARM Opcodes, Prev: ARM Floating Point, Up: ARM-Dependent

9.4.4 ARM Machine Directives

'.2byte EXPRESSION [, EXPRESSION]*'

'.4byte EXPRESSION [, EXPRESSION]*'

'.8byte EXPRESSION [, EXPRESSION]*'

These directives write 2, 4 or 8 byte values to the output section.

'.align EXPRESSION [, EXPRESSION]'

This is the generic .ALIGN directive. For the ARM however if the first argument is zero (ie no alignment is needed) the assembler will behave as if the argument had been 2 (ie pad to the next four byte boundary). This is for compatibility with ARM's own assembler.

'.arch NAME'

Select the target architecture. Valid values for NAME are the same as for the '-march' commandline option.

Specifying '.arch' clears any previously selected architecture extensions.

'.arch_extension NAME'

Add or remove an architecture extension to the target architecture.

Valid values for NAME are the same as those accepted as architectural extensions by the '-mcpu' commandline option.

'`.arch_extension`' may be used multiple times to add or remove extensions incrementally to the architecture being compiled for.

- '`.arm`'
This performs the same action as `.CODE 32`.
- '`.bss`'
This directive switches to the '`.bss`' section.
- '`.cantunwind`'
Prevents unwinding through the current function. No personality routine or exception table data is required or permitted.
- '`.code [16|32]`'
This directive selects the instruction set being generated. The value 16 selects Thumb, with the value 32 selecting ARM.
- '`.cpu NAME`'
Select the target processor. Valid values for NAME are the same as for the '-mcpu' commandline option.

Specifying '`.cpu`' clears any previously selected architecture extensions.

```
'NAME .dn REGISTER NAME [.TYPE] [[INDEX]]'
'NAME .qn REGISTER NAME [.TYPE] [[INDEX]]'
```

The '`dn`' and '`qn`' directives are used to create typed and/or indexed register aliases for use in Advanced SIMD Extension (Neon) instructions. The former should be used to create aliases of double-precision registers, and the latter to create aliases of quad-precision registers.

If these directives are used to create typed aliases, those aliases can be used in Neon instructions instead of writing types after the mnemonic or after each operand. For example:

```
x .dn d2.f32
y .dn d3.f32
z .dn d4.f32[1]
vmul x,y,z
```

This is equivalent to writing the following:

```
vmul.f32 d2,d3,d4[1]
```

Aliases created using '`dn`' or '`qn`' can be destroyed using '`unreq`'.

```
'.eabi_attribute TAG, VALUE'  
Set the EABI object attribute TAG to VALUE.
```

The TAG is either an attribute number, or one of the following:
'`Tag_CPU_raw_name`', '`Tag_CPU_name`', '`Tag_CPU_arch`',
'`Tag_CPU_arch_profile`', '`Tag_ARM_ISA_use`', '`Tag_THUMB_ISA_use`',
'`Tag_FP_arch`', '`Tag_WMMX_arch`', '`Tag_Advanced_SIMD_arch`',
'`Tag_PCS_config`', '`Tag_ABI_PCS_R9_use`', '`Tag_ABI_PCS_RW_data`',

'Tag_ABI_PCS_RO_data', 'Tag_ABI_PCS_GOT_use',
 'Tag_ABI_PCS_wchar_t', 'Tag_ABI_FP_rounding',
 'Tag_ABI_FP_denormal', 'Tag_ABI_FP_exceptions',
 'Tag_ABI_FP_user_exceptions', 'Tag_ABI_FP_number_model',
 'Tag_ABI_align_needed', 'Tag_ABI_align_preserved',
 'Tag_ABI_enum_size', 'Tag_ABI_HardFP_use', 'Tag_ABI_VFP_args',
 'Tag_ABI_WMMX_args', 'Tag_ABI_optimization_goals',
 'Tag_ABI_FP_optimization_goals', 'Tag_compatibility',
 'Tag_CPU_unaligned_access', 'Tag_FP_HP_extension',
 'Tag_ABI_FP_16bit_format', 'Tag_MPextension_use', 'Tag_DIV_use',
 'Tag_nodfaults', 'Tag_also_compatible_with', 'Tag_conformance',
 'Tag_T2EE_use', 'Tag_Virtualization_use'

The VALUE is either a 'number', '"string"', or 'number, "string"' depending on the tag.

Note - the following legacy values are also accepted by TAG:

'Tag_VFP_arch', 'Tag_ABI_align8_needed',
 'Tag_ABI_align8_preserved', 'Tag_VFP_HP_extension',

'.even'

This directive aligns to an even-numbered address.

'.extend EXPRESSION [, EXPRESSION]*'

'.ldouble EXPRESSION [, EXPRESSION]*'

These directives write 12byte long double floating-point values to the output section. These are not compatible with current ARM processors or ABIs.

'.fnend'

Marks the end of a function with an unwind table entry. The unwind index table entry is created when this directive is processed.

If no personality routine has been specified then standard personality routine 0 or 1 will be used, depending on the number of unwind opcodes required.

'.fnstart'

Marks the start of a function with an unwind table entry.

'.force_thumb'

This directive forces the selection of Thumb instructions, even if the target processor does not support those instructions

'.fpu NAME'

Select the floating-point unit to assemble for. Valid values for NAME are the same as for the '-mfpu' commandline option.

'.handlerdata'

Marks the end of the current function, and the start of the exception table entry for that function. Anything between this directive and the '.fnend' directive will be added to the exception table entry.

Must be preceded by a '.personality' or '.personalityindex' directive.

'.inst OPCODE [, ...]'

'.inst.n OPCODE [, ...]'

`'.inst.w OP CODE [, ...]'`
Generates the instruction corresponding to the numerical value OP CODE. `'.inst.n'` and `'.inst.w'` allow the Thumb instruction size to be specified explicitly, overriding the normal encoding rules.

`'.ldouble EXPRESSION [, EXPRESSION]*'`
See `'.extend'`.

`'.ltorg'`
This directive causes the current contents of the literal pool to be dumped into the current section (which is assumed to be the `.text` section) at the current location (aligned to a word boundary). `'GAS'` maintains a separate literal pool for each section and each sub-section. The `'.ltorg'` directive will only affect the literal pool of the current section and sub-section. At the end of assembly all remaining, un-empty literal pools will automatically be dumped.

Note - older versions of `'GAS'` would dump the current literal pool any time a section change occurred. This is no longer done, since it prevents accurate control of the placement of literal pools.

`'.movsp REG [, #OFFSET]'`
Tell the unwinder that REG contains an offset from the current stack pointer. If OFFSET is not specified then it is assumed to be zero.

`'.object_arch NAME'`
Override the architecture recorded in the EABI object attribute section. Valid values for NAME are the same as for the `'.arch'` directive. Typically this is useful when code uses runtime detection of CPU features.

`'.packed EXPRESSION [, EXPRESSION]*'`
This directive writes 12-byte packed floating-point values to the output section. These are not compatible with current ARM processors or ABIs.

`'.pad #COUNT'`
Generate unwinder annotations for a stack adjustment of COUNT bytes. A positive value indicates the function prologue allocated stack space by decrementing the stack pointer.

`'.personality NAME'`
Sets the personality routine for the current function to NAME.

`'.personalityindex INDEX'`
Sets the personality routine for the current function to the EABI standard routine number INDEX

`'.pool'`
This is a synonym for `.ltorg`.

`'NAME .req REGISTER NAME'`
This creates an alias for REGISTER NAME called NAME. For example:

`foo .req r0`

`'.save REGLIST'`

Generate unwinder annotations to restore the registers in REGLIST. The format of REGLIST is the same as the corresponding store-multiple instruction.

```
_core registers_
    .save {r4, r5, r6, lr}
    stmfid sp!, {r4, r5, r6, lr}
_FPA registers_
    .save f4, 2
    sfmfid f4, 2, [sp]!
_VFP registers_
    .save {d8, d9, d10}
    fstmdx sp!, {d8, d9, d10}
_iWMMXt registers_
    .save {wr10, wr11}
    wstrd wr11, [sp, #-8]!
    wstrd wr10, [sp, #-8]!
or
    .save wr11
    wstrd wr11, [sp, #-8]!
    .save wr10
    wstrd wr10, [sp, #-8]!
```

'`.setfp FPREG, SPREG [, #OFFSET]`'

Make all unwinder annotations relative to a frame pointer. Without this the unwinder will use offsets from the stack pointer.

The syntax of this directive is the same as the 'add' or 'mov' instruction used to set the frame pointer. SPREG must be either 'sp' or mentioned in a previous '`.movsp`' directive.

```
.movsp ip
mov ip, sp
...
.setfp fp, ip, #4
add fp, ip, #4
```

'`.secrel32 EXPRESSION [, EXPRESSION]*`'

This directive emits relocations that evaluate to the section-relative offset of each expression's symbol. This directive is only supported for PE targets.

'`.syntax [unified | divided]`'

This directive sets the Instruction Set Syntax as described in the *note ARM-Instruction-Set:: section.

'`.thumb`'

This performs the same action as `.CODE 16`.

'`.thumb_func`'

This directive specifies that the following symbol is the name of a Thumb encoded function. This information is necessary in order to allow the assembler and linker to generate correct code for interworking between Arm and Thumb instructions and should be used even if interworking is not going to be performed. The presence of this directive also implies '`.thumb`'

This directive is not necessary when generating EABI objects. On these targets the encoding is implicit when generating Thumb code.

'thumb_set'

This performs the equivalent of a `'.set'` directive in that it creates a symbol which is an alias for another symbol (possibly not yet defined). This directive also has the added property in that it marks the aliased symbol as being a thumb function entry point, in the same way that the `'.thumb_func'` directive does.

'tlsdescseq TLS-VARIABLE'

This directive is used to annotate parts of an inlined TLS descriptor trampoline. Normally the trampoline is provided by the linker, and this directive is not needed.

'unreq ALIAS-NAME'

This undefines a register alias which was previously defined using the `'req'`, `'dn'` or `'qn'` directives. For example:

```
foo .req r0
.unreq foo
```

An error occurs if the name is undefined. Note - this pseudo op can be used to delete builtin in register name aliases (eg `'r0'`). This should only be done if it is really necessary.

'unwind_raw OFFSET, BYTE1, ...'

Insert one of more arbitrary unwind opcode bytes, which are known to adjust the stack pointer by `OFFSET` bytes.

For example `'.unwind_raw 4, 0xb1, 0x01'` is equivalent to `'.save {r0}'`

'vsave VFP-REGLIST'

Generate unwinder annotations to restore the VFP registers in `VFP-REGLIST` using `FLDMD`. Also works for VFPv3 registers that are to be restored using `VLDM`. The format of `VFP-REGLIST` is the same as the corresponding store-multiple instruction.

```
_VFP registers_
    .vsave {d8, d9, d10}
    fstmdd sp!, {d8, d9, d10}
_VFPv3 registers_
    .vsave {d15, d16, d17}
    vstm sp!, {d15, d16, d17}
```

Since `FLDMX` and `FSTMX` are now deprecated, this directive should be used in favour of `'.save'` for saving VFP registers for ARMv6 and above.



File: as.info, Node: ARM Opcodes, Next: ARM Mapping Symbols, Prev: ARM Directives, Up: ARM-Dependent

9.4.5 Opcodes

`'as'` implements all the standard ARM opcodes. It also implements several pseudo opcodes, including several synthetic load instructions.

'NOP'

`nop`

This pseudo op will always evaluate to a legal ARM instruction that does nothing. Currently it will evaluate to `MOV r0, r0`.

'LDR'

```
ldr <register> , = <expression>
```

If expression evaluates to a numeric constant then a `MOV` or `MVN` instruction will be used in place of the `LDR` instruction, if the constant can be generated by either of these instructions. Otherwise the constant will be placed into the nearest literal pool (if it not already there) and a PC relative `LDR` instruction will be generated.

'ADR'

```
adr <register> <label>
```

This instruction will load the address of `LABEL` into the indicated register. The instruction will evaluate to a PC relative `ADD` or `SUB` instruction depending upon where the label is located. If the label is out of range, or if it is not defined in the same file (and section) as the `ADR` instruction, then an error will be generated. This instruction will not make use of the literal pool.

'ADRL'

```
adrl <register> <label>
```

This instruction will load the address of `LABEL` into the indicated register. The instruction will evaluate to one or two PC relative `ADD` or `SUB` instructions depending upon where the label is located. If a second instruction is not needed a `NOP` instruction will be generated in its place, so that this instruction is always 8 bytes long.

If the label is out of range, or if it is not defined in the same file (and section) as the `ADRL` instruction, then an error will be generated. This instruction will not make use of the literal pool.

For information on the ARM or Thumb instruction sets, see 'ARM Software Development Toolkit Reference Manual', Advanced RISC Machines Ltd.



File: as.info, Node: ARM Mapping Symbols, Next: ARM Unwinding Tutorial, Prev: ARM Opcodes, Up: ARM-Dependent

9.4.6 Mapping Symbols

The ARM ELF specification requires that special symbols be inserted into object files to mark certain features:

'\$a'

At the start of a region of code containing ARM instructions.

'\$t'

At the start of a region of code containing THUMB instructions.

'\$d'

At the start of a region of data.

The assembler will automatically insert these symbols for you - there is no need to code them yourself. Support for tagging symbols (\$b, \$f, \$p and \$m) which is also mentioned in the current ARM ELF specification is not implemented. This is because they have been dropped from the new EABI and so tools cannot rely upon their presence.

US

File: as.info, Node: ARM Unwinding Tutorial, Prev: ARM Mapping Symbols, Up: ARM-Dependent

9.4.7 Unwinding

The ABI for the ARM Architecture specifies a standard format for exception unwind information. This information is used when an exception is thrown to determine where control should be transferred. In particular, the unwind information is used to determine which function called the function that threw the exception, and which function called that one, and so forth. This information is also used to restore the values of callee-saved registers in the function catching the exception.

If you are writing functions in assembly code, and those functions call other functions that throw exceptions, you must use assembly pseudo ops to ensure that appropriate exception unwind information is generated. Otherwise, if one of the functions called by your assembly code throws an exception, the run-time library will be unable to unwind the stack through your assembly code and your program will not behave correctly.

To illustrate the use of these pseudo ops, we will examine the code that G++ generates for the following C++ input:

```
void callee (int *);

int
caller ()
{
    int i;
    callee (&i);
    return i;
}
```

This example does not show how to throw or catch an exception from assembly code. That is a much more complex operation and should always be done in a high-level language, such as C++, that directly supports exceptions.

The code generated by one particular version of G++ when compiling the example above is:

```
_Z6callerv:
    .fnstart
.LFB2:
    @ Function supports interworking.
    @ args = 0, pretend = 0, frame = 8
```

```
@ frame_needed = 1, uses_anonymous_args = 0
stmfd sp!, {fp, lr}
.save {fp, lr}
.LCFI0:
.setfp fp, sp, #4
add fp, sp, #4
.LCFI1:
.pad #8
sub sp, sp, #8
.LCFI2:
sub r3, fp, #8
mov r0, r3
bl _Z6calleePi
ldr r3, [fp, #-8]
mov r0, r3
sub sp, fp, #4
ldmfd sp!, {fp, lr}
bx lr
.LFE2:
.fnend
```

Of course, the sequence of instructions varies based on the options you pass to GCC and on the version of GCC in use. The exact instructions are not important since we are focusing on the pseudo ops that are used to generate unwind information.

An important assumption made by the unwinder is that the stack frame does not change during the body of the function. In particular, since we assume that the assembly code does not itself throw an exception, the only point where an exception can be thrown is from a call, such as the 'bl' instruction above. At each call site, the same saved registers (including 'lr', which indicates the return address) must be located in the same locations relative to the frame pointer.

The '.fnstart' (*note .fnstart pseudo op: arm_fnstart.) pseudo op appears immediately before the first instruction of the function while the '.fnend' (*note .fnend pseudo op: arm_fnend.) pseudo op appears immediately after the last instruction of the function. These pseudo ops specify the range of the function.

Only the order of the other pseudos ops (e.g., '.setfp' or '.pad') matters; their exact locations are irrelevant. In the example above, the compiler emits the pseudo ops with particular instructions. That makes it easier to understand the code, but it is not required for correctness. It would work just as well to emit all of the pseudo ops other than '.fnend' in the same order, but immediately after '.fnstart'.

The '.save' (*note .save pseudo op: arm_save.) pseudo op indicates registers that have been saved to the stack so that they can be restored before the function returns. The argument to the '.save' pseudo op is a list of registers to save. If a register is "callee-saved" (as specified by the ABI) and is modified by the function you are writing, then your code must save the value before it is modified and restore the original value before the function returns. If an exception is thrown, the run-time library restores the values of these registers from their locations on the stack before returning control to the exception handler. (Of course, if an exception is not thrown, the function that contains the '.save' pseudo op restores these registers in the function epilogue, as is done with the 'ldmfd' instruction above.)

You do not have to save callee-saved registers at the very beginning of the function and you do not need to use the `'.save'` pseudo op immediately following the point at which the registers are saved. However, if you modify a callee-saved register, you must save it on the stack before modifying it and before calling any functions which might throw an exception. And, you must use the `'.save'` pseudo op to indicate that you have done so.

The `'.pad'` (*note `.pad: arm_pad.`) pseudo op indicates a modification of the stack pointer that does not save any registers. The argument is the number of bytes (in decimal) that are subtracted from the stack pointer. (On ARM CPUs, the stack grows downwards, so subtracting from the stack pointer increases the size of the stack.)

The `'.setfp'` (*note `.setfp pseudo op: arm_setfp.`) pseudo op indicates the register that contains the frame pointer. The first argument is the register that is set, which is typically `'fp'`. The second argument indicates the register from which the frame pointer takes its value. The third argument, if present, is the value (in decimal) added to the register specified by the second argument to compute the value of the frame pointer. You should not modify the frame pointer in the body of the function.

If you do not use a frame pointer, then you should not use the `'.setfp'` pseudo op. If you do not use a frame pointer, then you should avoid modifying the stack pointer outside of the function prologue. Otherwise, the run-time library will be unable to find saved registers when it is unwinding the stack.

The pseudo ops described above are sufficient for writing assembly code that calls functions which may throw exceptions. If you need to know more about the object-file format used to represent unwind information, you may consult the 'Exception Handling ABI for the ARM Architecture' available from <http://infocenter.arm.com>.



File: as.info, Node: AVR-Dependent, Next: Blackfin-Dependent, Prev: ARM-Dependent, Up: Machine Dependencies

9.5 AVR Dependent Features

=====

* Menu:

* AVR Options::	Options
* AVR Syntax::	Syntax
* AVR Opcodes::	Opcodes



File: as.info, Node: AVR Options, Next: AVR Syntax, Up: AVR-Dependent

9.5.1 Options

'-mmcuc=MCU'

Specify ATMEL AVR instruction set or MCU type.

Instruction set `avr1` is for the minimal AVR core, not supported by

the C compiler, only for assembler programs (MCU types: at90s1200, attiny11, attiny12, attiny15, attiny28).

Instruction set avr2 (default) is for the classic AVR core with up to 8K program memory space (MCU types: at90s2313, at90s2323, at90s2333, at90s2343, attiny22, attiny26, at90s4414, at90s4433, at90s4434, at90s8515, at90c8534, at90s8535).

Instruction set avr25 is for the classic AVR core with up to 8K program memory space plus the MOVW instruction (MCU types: attiny13, attiny13a, attiny2313, attiny2313a, attiny24, attiny24a, attiny4313, attiny44, attiny44a, attiny84, attiny84a, attiny25, attiny45, attiny85, attiny261, attiny261a, attiny461, attiny461a, attiny861, attiny861a, attiny87, attiny43u, attiny48, attiny88, attiny828, at86rf401, ata6289, ata5272).

Instruction set avr3 is for the classic AVR core with up to 128K program memory space (MCU types: at43usb355, at76c711).

Instruction set avr31 is for the classic AVR core with exactly 128K program memory space (MCU types: atmega103, at43usb320).

Instruction set avr35 is for classic AVR core plus MOVW, CALL, and JMP instructions (MCU types: attiny167, attiny1634, at90usb82, at90usb162, atmega8u2, atmega16u2, atmega32u2, ata5505).

Instruction set avr4 is for the enhanced AVR core with up to 8K program memory space (MCU types: atmega48, atmega48a, atmega48pa, atmega48p, atmega8, atmega8a, atmega88, atmega88a, atmega88p, atmega88pa, atmega8515, atmega8535, atmega8hva, at90pwm1, at90pwm2, at90pwm2b, at90pwm3, at90pwm3b, at90pwm81, ata6285, ata6286).

Instruction set avr5 is for the enhanced AVR core with up to 128K program memory space (MCU types: at90pwm161, atmega16, atmega16a, atmega161, atmega162, atmega163, atmega164a, atmega164p, atmega164pa, atmega165, atmega165a, atmega165p, atmega165pa, atmega168, atmega168a, atmega168p, atmega168pa, atmega169, atmega169a, atmega169p, atmega169pa, atmega32, atmega323, atmega324a, atmega324p, atmega324pa, atmega325, atmega325a, atmega32, atmega32a, atmega323, atmega324a, atmega324p, atmega324pa, atmega325, atmega325a, atmega325p, atmega325p, atmega325pa, atmega3250, atmega3250a, atmega3250p, atmega3250pa, atmega328, atmega328p, atmega329, atmega329a, atmega329p, atmega329pa, atmega3290a, atmega3290p, atmega3290pa, atmega406, atmega64, atmega64a, atmega64rfr2, atmega644rfr2, atmega640, atmega644, atmega644a, atmega644p, atmega644pa, atmega645, atmega645a, atmega645p, atmega6450, atmega6450a, atmega6450p, atmega649, atmega649a, atmega649p, atmega6490, atmega6490a, atmega6490p, atmega16hva, atmega16hva2, atmega16hvb, atmega16hvbrevb, atmega32hvb, atmega32hvbrevb, atmega64hve, at90can32, at90can64, at90pwm161, at90pwm216, at90pwm316, atmega32c1, atmega64c1, atmega16m1, atmega32m1, atmega64m1, atmega16u4, atmega32u4, atmega32u6, at90usb646, at90usb647, at94k, at90scr100, ata5790, ata5795).

Instruction set avr51 is for the enhanced AVR core with exactly 128K program memory space (MCU types: atmega128, atmega128a, atmega1280, atmega1281, atmega1284, atmega1284p, atmega128rfa1, atmega128rfr2, atmega1284rfr2, at90can128, at90usb1286,

at90usb1287, m3000).

Instruction set avr6 is for the enhanced AVR core with a 3-byte PC (MCU types: atmega2560, atmega2561, atmega256rfr2, atmega2564rfr2).

Instruction set avrxmega2 is for the XMEGA AVR core with 8K to 64K program memory space and less than 64K data space (MCU types: atxmega16a4, atxmega16a4u, atxmega16c4, atxmega16d4, atxmega16x1, atxmega32a4, atxmega32a4u, atxmega32c4, atxmega32d4, atxmega16e5, atxmega8e5, atxmega32e5, atxmega32x1).

Instruction set avrxmega3 is for the XMEGA AVR core with 8K to 64K program memory space and greater than 64K data space (MCU types: none).

Instruction set avrxmega4 is for the XMEGA AVR core with up to 64K program memory space and less than 64K data space (MCU types: atxmega64a3, atxmega64a3u, atxmega64a4u, atxmega64b1, atxmega64b3, atxmega64c3, atxmega64d3, atxmega64d4).

Instruction set avrxmega5 is for the XMEGA AVR core with up to 64K program memory space and greater than 64K data space (MCU types: atxmega64a1, atxmega64a1u).

Instruction set avrxmega6 is for the XMEGA AVR core with larger than 64K program memory space and less than 64K data space (MCU types: atxmega128a3, atxmega128a3u, atxmega128c3, atxmega128d3, atxmega128d4, atxmega192a3, atxmega192a3u, atxmega128b1, atxmega128b3, atxmega192c3, atxmega192d3, atxmega256a3, atxmega256a3u, atxmega256a3b, atxmega256a3bu, atxmega256c3, atxmega256d3, atxmega384c3, atxmega256d3).

Instruction set avrxmega7 is for the XMEGA AVR core with larger than 64K program memory space and greater than 64K data space (MCU types: atxmega128a1, atxmega128a1u, atxmega128a4u).

Instruction set avrtiny is for the ATtiny4/5/9/10/20/40 microcontrollers.

'-mall-opcodes'

Accept all AVR opcodes, even if not supported by '-mmcu'.

'-mno-skip-bug'

This option disable warnings for skipping two-word instructions.

'-mno-wrap'

This option reject 'rjmp/rcall' instructions with 8K wrap-around.

'-mrmw'

Accept Read-Modify-Write ('XCH,LAC,LAS,LAT') instructions.

'-mlink-relax'

Enable support for link-time relaxation. This is now on by default and this flag no longer has any effect.

'-mno-link-relax'

Disable support for link-time relaxation. The assembler will resolve relocations when it can, and may be able to better compress some debug information.



File: as.info, Node: AVR Syntax, Next: AVR Opcodes, Prev: AVR Options, Up: AVR-Dependent

9.5.2 Syntax

* Menu:

* AVR-Chars:: Special Characters
 * AVR-Regs:: Register Names
 * AVR-Modifiers:: Relocatable Expression Modifiers



File: as.info, Node: AVR-Chars, Next: AVR-Regs, Up: AVR Syntax

9.5.2.1 Special Characters

.....

The presence of a ';' anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The '\$' character can be used instead of a newline to separate statements.



File: as.info, Node: AVR-Regs, Next: AVR-Modifiers, Prev: AVR-Chars, Up: AVR Syntax

9.5.2.2 Register Names

.....

The AVR has 32 x 8-bit general purpose working registers 'r0', 'r1', ... 'r31'. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing. One of these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit 'X', 'Y' and 'Z' - registers.

```
X = r26:r27
Y = r28:r29
Z = r30:r31
```



File: as.info, Node: AVR-Modifiers, Prev: AVR-Regs, Up: AVR Syntax

9.5.2.3 Relocatable Expression Modifiers

.....

The assembler supports several modifiers when using relocatable addresses in AVR instruction operands. The general syntax is the following:

modifier(relocatable-expression)

'lo8'

This modifier allows you to use bits 0 through 7 of an address expression as 8 bit relocatable expression.

'hi8'

This modifier allows you to use bits 7 through 15 of an address expression as 8 bit relocatable expression. This is useful with, for example, the AVR 'ldi' instruction and 'lo8' modifier.

For example

```
ldi r26, lo8(sym+10)
ldi r27, hi8(sym+10)
```

'hh8'

This modifier allows you to use bits 16 through 23 of an address expression as 8 bit relocatable expression. Also, can be useful for loading 32 bit constants.

'hlo8'

Synonym of 'hh8'.

'hhi8'

This modifier allows you to use bits 24 through 31 of an expression as 8 bit expression. This is useful with, for example, the AVR 'ldi' instruction and 'lo8', 'hi8', 'hlo8', 'hhi8', modifier.

For example

```
ldi r26, lo8(285774925)
ldi r27, hi8(285774925)
ldi r28, hlo8(285774925)
ldi r29, hhi8(285774925)
; r29,r28,r27,r26 = 285774925
```

'pm_lo8'

This modifier allows you to use bits 0 through 7 of an address expression as 8 bit relocatable expression. This modifier useful for addressing data or code from Flash/Program memory. The using of 'pm_lo8' similar to 'lo8'.

'pm_hi8'

This modifier allows you to use bits 8 through 15 of an address expression as 8 bit relocatable expression. This modifier useful for addressing data or code from Flash/Program memory.

'pm_hh8'

This modifier allows you to use bits 15 through 23 of an address expression as 8 bit relocatable expression. This modifier useful

for addressing data or code from Flash/Program memory.



File: as.info, Node: AVR Opcodes, Prev: AVR Syntax, Up: AVR-Dependent

9.5.3 Opcodes

For detailed information on the AVR machine instruction set, see www.atmel.com/products/AVR.

'as' implements all the standard AVR opcodes. The following table summarizes the AVR opcodes, and their arguments.

Legend:

r any register
d 'ldi' register (r16-r31)
v 'movw' even register (r0, r2, ..., r28, r30)
a 'fmul' register (r16-r23)
w 'adiw' register (r24,r26,r28,r30)
e pointer registers (X,Y,Z)
b base pointer register and displacement ([YZ]+disp)
z Z pointer register (for [e]lpm Rd,Z[+])
M immediate value from 0 to 255
n immediate value from 0 to 255 (n = ~M). Relocation impossible
s immediate value from 0 to 7
P Port address value from 0 to 63. (in, out)
p Port address value from 0 to 31. (cbi, sbi, sbic, sbis)
K immediate value from 0 to 63 (used in 'adiw', 'sbiw')
i immediate value
l signed pc relative offset from -64 to 63
L signed pc relative offset from -2048 to 2047
h absolute code address (call, jmp)
S immediate value from 0 to 7 (S = s << 4)
? use this opcode entry if no parameters, else use next opcode entry

1001010010001000	clc	
1001010011011000	clh	
1001010011111000	cli	
1001010010101000	cln	
1001010011001000	cls	
1001010011101000	clt	
1001010010111000	clv	
1001010010011000	clz	
1001010000001000	sec	
1001010001011000	seh	
1001010001111000	sei	
1001010000101000	sen	
1001010001001000	ses	
1001010001101000	set	
1001010000111000	sev	
1001010000011000	sez	
100101001SSS1000	bclr	S
100101000SSS1000	bset	S
1001010100001001	icall	
1001010000001001	ijmp	
1001010111001000	lpm	?
1001000dddd010+	lpm	r,z
1001010111011000	elpm	?

1001000dddd011+	elpm	r,z
0000000000000000	nop	
1001010100001000	ret	
1001010100011000	reti	
1001010110001000	sleep	
1001010110011000	break	
1001010110101000	wdr	
1001010111101000	spm	
000111rddddrrrr	adc	r,r
000011rddddrrrr	add	r,r
001000rddddrrrr	and	r,r
000101rddddrrrr	cp	r,r
000001rddddrrrr	cpc	r,r
000100rddddrrrr	cpse	r,r
001001rddddrrrr	eor	r,r
001011rddddrrrr	mov	r,r
100111rddddrrrr	mul	r,r
001010rddddrrrr	or	r,r
000010rddddrrrr	sbc	r,r
000110rddddrrrr	sub	r,r
001001rddddrrrr	clr	r
000011rddddrrrr	lsl	r
000111rddddrrrr	rol	r
001000rddddrrrr	tst	r
0111KKKKddddKKKK	andi	d,M
0111KKKKddddKKKK	cbr	d,n
1110KKKKddddKKKK	ldi	d,M
11101111dddd1111	ser	d
0110KKKKddddKKKK	ori	d,M
0110KKKKddddKKKK	sbr	d,M
0011KKKKddddKKKK	cpi	d,M
0100KKKKddddKKKK	sbc	d,M
0101KKKKddddKKKK	subi	d,M
1111110rrrrr0sss	sbr	r,s
1111111rrrrr0sss	sbrs	r,s
1111100dddd0sss	bld	r,s
1111101dddd0sss	bst	r,s
10110PPddddPPPP	in	r,P
10111PPrrrrrPPPP	out	P,r
10010110KKddKKKK	adiw	w,K
10010111KKddKKKK	sbiw	w,K
10011000pppppsss	cbi	p,s
10011010pppppsss	sbi	p,s
10011001pppppsss	sbic	p,s
10011011pppppsss	sbis	p,s
1111011111111000	brcc	l
1111001111111000	brcs	l
1111001111111001	breq	l
1111011111111100	brge	l
1111011111111101	brhc	l
1111001111111101	brhs	l
1111011111111111	brid	l
1111001111111111	brie	l
1111001111111000	brlo	l
1111001111111100	brlt	l
1111001111111010	brmi	l
1111011111111001	brne	l
1111011111111010	brpl	l
1111011111111000	brsh	l

```

1111011111111110   brtc   l
1111001111111110   brts   l
11110111111111011  brvc   l
11110011111111011  brvs   l
111101111111111ss  brbc   s,l
111100111111111ss  brbs   s,l
1101LLLLLLLLLLLLL  rcall  L
1100LLLLLLLLLLLLL  rjmp   L
1001010hhhhh111h   call   h
1001010hhhhh110h   jmp    h
1001010rrrrr0101   asr    r
1001010rrrrr0000   com    r
1001010rrrrr1010   dec    r
1001010rrrrr0011   inc    r
1001010rrrrr0110   lsr    r
1001010rrrrr0001   neg    r
1001000rrrrr1111   pop    r
1001001rrrrr1111   push   r
1001010rrrrr0111   ror    r
1001010rrrrr0010   swap   r
00000001ddddrrrr   movw   v,v
00000010ddddrrrr   muls   d,d
000000110ddd0rrr   mulsu  a,a
000000110ddd1rrr   fmul   a,a
000000111ddd0rrr   fmul   a,a
000000111ddd1rrr   fmulsu a,a
1001001dddd0000   sts    i,r
1001000dddd0000   lds    r,i
10o0oo0ddddbooo   ldd    r,b
100!000dddddee-+   ld     r,e
10o0oo1rrrrrbooo   std    b,r
100!001rrrrree-+   st     e,r
1001010100011001   eicall
1001010000011001   eijmp

```



File: as.info, Node: Blackfin-Dependent, Next: CR16-Dependent, Prev: AVR-Dependent, Up: Machine Dependencies

9.6 Blackfin Dependent Features

=====

* Menu:

- * Blackfin Options:: Blackfin Options
- * Blackfin Syntax:: Blackfin Syntax
- * Blackfin Directives:: Blackfin Directives



File: as.info, Node: Blackfin Options, Next: Blackfin Syntax, Up: Blackfin-Dependent

9.6.1 Options

'-mcpu=PROCESSOR[-SIREVISION]'

This option specifies the target processor. The optional SIREVISION is not used in assembler. It's here such that GCC can easily pass down its '-mcpu=' option. The assembler will issue an

error message if an attempt is made to assemble an instruction which will not execute on the target processor. The following processor names are recognized: 'bf504', 'bf506', 'bf512', 'bf514', 'bf516', 'bf518', 'bf522', 'bf523', 'bf524', 'bf525', 'bf526', 'bf527', 'bf531', 'bf532', 'bf533', 'bf534', 'bf535' (not implemented yet), 'bf536', 'bf537', 'bf538', 'bf539', 'bf542', 'bf542m', 'bf544', 'bf544m', 'bf547', 'bf547m', 'bf548', 'bf548m', 'bf549', 'bf549m', 'bf561', and 'bf592'.

'-mfdpic'

Assemble for the FDPIC ABI.

'-mno-fdpic'

'-mnopic'

Disable -mfdpic.



File: as.info, Node: Blackfin Syntax, Next: Blackfin Directives, Prev: Blackfin Options, Up: Blackfin-Dependent

9.6.2 Syntax

'Special Characters'

Assembler input is free format and may appear anywhere on the line. One instruction may extend across multiple lines or more than one instruction may appear on the same line. White space (space, tab, comments or newline) may appear anywhere between tokens. A token must not have embedded spaces. Tokens include numbers, register names, keywords, user identifiers, and also some multicharacter special symbols like "+=", "/*" or "||".

Comments are introduced by the '#' character and extend to the end of the current line. If the '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

'Instruction Delimiting'

A semicolon must terminate every instruction. Sometimes a complete instruction will consist of more than one operation. There are two cases where this occurs. The first is when two general operations are combined. Normally a comma separates the different parts, as in

```
a0= r3.h * r2.l, a1 = r3.l * r2.h ;
```

The second case occurs when a general instruction is combined with one or two memory references for joint issue. The latter portions are set off by a "||" token.

```
a0 = r3.h * r2.l || r1 = [p3++] || r4 = [i2++];
```

Multiple instructions can occur on the same line. Each must be terminated by a semicolon character.

'Register Names'

The assembler treats register names and instruction keywords in a case insensitive manner. User identifiers are case sensitive. Thus, R3.l, R3.L, r3.l and r3.L are all equivalent input to the assembler.

Register names are reserved and may not be used as program identifiers.

Some operations (such as "Move Register") require a register pair. Register pairs are always data registers and are denoted using a colon, eg., R3:2. The larger number must be written first. Note that the hardware only supports odd-even pairs, eg., R7:6, R5:4, R3:2, and R1:0.

Some instructions (such as -SP (Push Multiple)) require a group of adjacent registers. Adjacent registers are denoted in the syntax by the range enclosed in parentheses and separated by a colon, eg., (R7:3). Again, the larger number appears first.

Portions of a particular register may be individually specified. This is written with a dot (".") following the register name and then a letter denoting the desired portion. For 32-bit registers, ".H" denotes the most significant ("High") portion. ".L" denotes the least-significant portion. The subdivisions of the 40-bit registers are described later.

'Accumulators'

The set of 40-bit registers A1 and A0 that normally contain data that is being manipulated. Each accumulator can be accessed in four ways.

'one 40-bit register'

The register will be referred to as A1 or A0.

'one 32-bit register'

The registers are designated as A1.W or A0.W.

'two 16-bit registers'

The registers are designated as A1.H, A1.L, A0.H or A0.L.

'one 8-bit register'

The registers are designated as A1.X or A0.X for the bits that extend beyond bit 31.

'Data Registers'

The set of 32-bit registers (R0, R1, R2, R3, R4, R5, R6 and R7) that normally contain data for manipulation. These are abbreviated as D-register or Dreg. Data registers can be accessed as 32-bit registers or as two independent 16-bit registers. The least significant 16 bits of each register is called the "low" half and is designated with ".L" following the register name. The most significant 16 bits are called the "high" half and is designated with ".H" following the name.

R7.L, r2.h, r4.L, R0.H

'Pointer Registers'

The set of 32-bit registers (P0, P1, P2, P3, P4, P5, SP and FP) that normally contain byte addresses of data structures. These are abbreviated as P-register or Preg.

p2, p5, fp, sp

'Stack Pointer SP'

The stack pointer contains the 32-bit address of the last occupied byte location in the stack. The stack grows by decrementing the stack pointer.

'Frame Pointer FP'

The frame pointer contains the 32-bit address of the previous frame pointer in the stack. It is located at the top of a frame.

'Loop Top'

LT0 and LT1. These registers contain the 32-bit address of the top of a zero overhead loop.

'Loop Count'

LC0 and LC1. These registers contain the 32-bit counter of the zero overhead loop executions.

'Loop Bottom'

LB0 and LB1. These registers contain the 32-bit address of the bottom of a zero overhead loop.

'Index Registers'

The set of 32-bit registers (I0, I1, I2, I3) that normally contain byte addresses of data structures. Abbreviated I-register or Ireg.

'Modify Registers'

The set of 32-bit registers (M0, M1, M2, M3) that normally contain offset values that are added and subtracted to one of the index registers. Abbreviated as Mreg.

'Length Registers'

The set of 32-bit registers (L0, L1, L2, L3) that normally contain the length in bytes of the circular buffer. Abbreviated as Lreg. Clear the Lreg to disable circular addressing for the corresponding Ireg.

'Base Registers'

The set of 32-bit registers (B0, B1, B2, B3) that normally contain the base address in bytes of the circular buffer. Abbreviated as Breg.

'Floating Point'

The Blackfin family has no hardware floating point but the .float directive generates ieee floating point numbers for use with software floating point libraries.

'Blackfin Opcodes'

For detailed information on the Blackfin machine instruction set, see the Blackfin(r) Processor Instruction Set Reference.



File: as.info, Node: Blackfin Directives, Prev: Blackfin Syntax, Up: Blackfin-Dependent

9.6.3 Directives

The following directives are provided for compatibility with the VDSP

assembler.

`'.byte2'`
 Initializes a two byte data object.
 This maps to the `'.short'` directive.

`'.byte4'`
 Initializes a four byte data object.
 This maps to the `'.int'` directive.

`'.db'`
 Initializes a single byte data object.
 This directive is a synonym for `'.byte'`.

`'.dw'`
 Initializes a two byte data object.
 This directive is a synonym for `'.byte2'`.

`'.dd'`
 Initializes a four byte data object.
 This directive is a synonym for `'.byte4'`.

`'.var'`
 Define and initialize a 32 bit data object.



File: as.info, Node: CR16-Dependent, Next: CRIS-Dependent, Prev: Blackfin-Dependent, Up: Machine Dependencies

9.7 CR16 Dependent Features

=====

* Menu:

* CR16 Operand Qualifiers:: CR16 Machine Operand Qualifiers
 * CR16 Syntax:: Syntax for the CR16



File: as.info, Node: CR16 Operand Qualifiers, Next: CR16 Syntax, Up: CR16-Dependent

9.7.1 CR16 Operand Qualifiers

The National Semiconductor CR16 target of 'as' has a few machine dependent operand qualifiers.

Operand expression type qualifier is an optional field in the instruction operand, to determines the type of the expression field of an operand. The '@' is required. CR16 architecture uses one of the following expression qualifiers:

`'s'`
 - 'Specifies expression operand type as small'

`'m'`
 - 'Specifies expression operand type as medium'

`'l'`
 - 'Specifies expression operand type as large'

`'c'`

- 'Specifies the CR16 Assembler generates a relocation entry for the operand, where pc has implied bit, the expression is adjusted accordingly. The linker uses the relocation entry to update the operand address at link time.'

'got/GOT'

- 'Specifies the CR16 Assembler generates a relocation entry for the operand, offset from Global Offset Table. The linker uses this relocation entry to update the operand address at link time'

'cgot/cGOT'

- 'Specifies the CompactRISC Assembler generates a relocation entry for the operand, where pc has implied bit, the expression is adjusted accordingly. The linker uses the relocation entry to update the operand address at link time.'

CR16 target operand qualifiers and its size (in bits):

'Immediate Operand: s'
4 bits.

'Immediate Operand: m'
16 bits, for movb and movw instructions.

'Immediate Operand: m'
20 bits, movd instructions.

'Immediate Operand: l'
32 bits.

'Absolute Operand: s'
Illegal specifier for this operand.

'Absolute Operand: m'
20 bits, movd instructions.

'Displacement Operand: s'
8 bits.

'Displacement Operand: m'
16 bits.

'Displacement Operand: l'
24 bits.

For example:

```
1  movw $_myfun@c,r1
```

This loads the address of `_myfun`, shifted right by 1, into r1.

```
2  movd $_myfun@c,(r2,r1)
```

This loads the address of `_myfun`, shifted right by 1, into register-pair r2-r1.

```
3  _myfun_ptr:
   .long _myfun@c
   loadd _myfun_ptr, (r1,r0)
   jal (r1,r0)
```

This `.long` directive, the address of `_myfunc`, shifted right by 1 at link

This .long directive, the address of `_myfunc`, shifted right by 1 at link time.

```
4  loadl  _data1@GOT(r12), (r1,r0)
```

This loads the address of `_data1`, into global offset table (ie GOT) and its offset value from GOT loads into register-pair r2-r1.

```
5  loadl  _myfunc@cGOT(r12), (r1,r0)
```

This loads the address of `_myfun`, shifted right by 1, into global offset table (ie GOT) and its offset value from GOT loads into register-pair r1-r0.

US

File: as.info, Node: CR16 Syntax, Prev: CR16 Operand Qualifiers, Up: CR16-Dependent

9.7.2 CR16 Syntax

* Menu:

* CR16-Chars:: Special Characters

US

File: as.info, Node: CR16-Chars, Up: CR16 Syntax

9.7.2.1 Special Characters

.....

The presence of a '#' on a line indicates the start of a comment that extends to the end of the current line. If the '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (`*note Comments::`) or a preprocessor control command (`*note Preprocessing::`).

The ';' character can be used to separate statements on the same line.

US

File: as.info, Node: CRIS-Dependent, Next: D10V-Dependent, Prev: CR16-Dependent, Up: Machine Dependencies

9.8 CRIS Dependent Features

=====

* Menu:

* CRIS-Opts:: Command-line Options
 * CRIS-Expand:: Instruction expansion
 * CRIS-Symbols:: Symbols
 * CRIS-Syntax:: Syntax

US

File: as.info, Node: CRIS-Opts, Next: CRIS-Expand, Up: CRIS-Dependent

9.8.1 Command-line Options

The CRIS version of 'as' has these machine-dependent command-line options.

The format of the generated object files can be either ELF or a.out, specified by the command-line options '--emulation=crisaout' and '--emulation=criself'. The default is ELF (criself), unless 'as' has been configured specifically for a.out by using the configuration name 'cris-axis-aout'.

There are two different link-incompatible ELF object file variants for CRIS, for use in environments where symbols are expected to be prefixed by a leading '_' character and for environments without such a symbol prefix. The variant used for GNU/Linux port has no symbol prefix. Which variant to produce is specified by either of the options '--underscore' and '--no-underscore'. The default is '--underscore'. Since symbols in CRIS a.out objects are expected to have a '_' prefix, specifying '--no-underscore' when generating a.out objects is an error. Besides the object format difference, the effect of this option is to parse register names differently (*note crisnous:). The '--no-underscore' option makes a '\$' register prefix mandatory.

The option '--pic' must be passed to 'as' in order to recognize the symbol syntax used for ELF (SVR4 PIC) position-independent-code (*note crispic:). This will also affect expansion of instructions. The expansion with '--pic' will use PC-relative rather than (slightly faster) absolute addresses in those expansions. This option is only valid when generating ELF format object files.

The option '--march=ARCHITECTURE' specifies the recognized instruction set and recognized register names. It also controls the architecture type of the object file. Valid values for ARCHITECTURE are:

'v0_v10'

All instructions and register names for any architecture variant in the set v0...v10 are recognized. This is the default if the target is configured as cris-*.

'v10'

Only instructions and register names for CRIS v10 (as found in ETRAX 100 LX) are recognized. This is the default if the target is configured as crisv10-*.

'v32'

Only instructions and register names for CRIS v32 (code name Guinness) are recognized. This is the default if the target is configured as crisv32-*. This value implies '--no-mul-bug-abort'. (A subsequent '--mul-bug-abort' will turn it back on.)

'common_v10_v32'

Only instructions with register names and addressing modes with opcodes common to the v10 and v32 are recognized.

When '-N' is specified, 'as' will emit a warning when a 16-bit branch instruction is expanded into a 32-bit multiple-instruction construct (*note CRIS-Expand:).

Some versions of the CRIS v10, for example in the Etrax 100 LX, contain a bug that causes destabilizing memory accesses when a multiply

instruction is executed with certain values in the first operand just before a cache-miss. When the '--mul-bug-abort' command line option is active (the default value), 'as' will refuse to assemble a file containing a multiply instruction at a dangerous offset, one that could be the last on a cache-line, or is in a section with insufficient alignment. This placement checking does not catch any case where the multiply instruction is dangerously placed because it is located in a delay-slot. The '--mul-bug-abort' command line option turns off the checking.

US

File: as.info, Node: CRIS-Expand, Next: CRIS-Symbols, Prev: CRIS-Opts, Up: CRIS-Dependent

9.8.2 Instruction expansion

'as' will silently choose an instruction that fits the operand size for '[register+constant]' operands. For example, the offset '127' in 'move.d [r3+127],r4' fits in an instruction using a signed-byte offset. Similarly, 'move.d [r2+32767],r1' will generate an instruction using a 16-bit offset. For symbolic expressions and constants that do not fit in 16 bits including the sign bit, a 32-bit offset is generated.

For branches, 'as' will expand from a 16-bit branch instruction into a sequence of instructions that can reach a full 32-bit address. Since this does not correspond to a single instruction, such expansions can optionally be warned about. *Note CRIS-Opts:..

If the operand is found to fit the range, a 'lapc' mnemonic will translate to a 'lapcq' instruction. Use 'lapc.d' to force the 32-bit 'lapc' instruction.

Similarly, the 'addo' mnemonic will translate to the shortest fitting instruction of 'addoq', 'addo.w' and 'addo.d', when used with a operand that is a constant known at assembly time.

US

File: as.info, Node: CRIS-Symbols, Next: CRIS-Syntax, Prev: CRIS-Expand, Up: CRIS-Dependent

9.8.3 Symbols

Some symbols are defined by the assembler. They're intended to be used in conditional assembly, for example:

```
.if ..asm.arch.cris.v32
CODE FOR CRIS V32
.elseif ..asm.arch.cris.common_v10_v32
CODE COMMON TO CRIS V32 AND CRIS V10
.elseif ..asm.arch.cris.v10 | ..asm.arch.cris.any_v0_v10
CODE FOR V10
.else
.error "Code needs to be added here."
.endif
```

These symbols are defined in the assembler, reflecting command-line options, either when specified or the default. They are always defined, to 0 or 1.

```
'..asm.arch.cris.any_v0_v10'
  This symbol is non-zero when '--march=v0_v10' is specified or the
  default.

'..asm.arch.cris.common_v10_v32'
  Set according to the option '--march=common_v10_v32'.

'..asm.arch.cris.v10'
  Reflects the option '--march=v10'.

'..asm.arch.cris.v32'
  Corresponds to '--march=v10'.
```

Speaking of symbols, when a symbol is used in code, it can have a suffix modifying its value for use in position-independent code. *Note CRIS-Pic::.



File: as.info, Node: CRIS-Syntax, Prev: CRIS-Symbols, Up: CRIS-Dependent

9.8.4 Syntax

There are different aspects of the CRIS assembly syntax.

* Menu:

```
* CRIS-Chars::          Special Characters
* CRIS-Pic::           Position-Independent Code Symbols
* CRIS-Regs::         Register Names
* CRIS-Pseudos::      Assembler Directives
```



File: as.info, Node: CRIS-Chars, Next: CRIS-Pic, Up: CRIS-Syntax

9.8.4.1 Special Characters

.....

The character '#' is a line comment character. It starts a comment if and only if it is placed at the beginning of a line.

A ';' character starts a comment anywhere on the line, causing all characters up to the end of the line to be ignored.

A '@' character is handled as a line separator equivalent to a logical new-line character (except in a comment), so separate instructions can be specified on a single line.



File: as.info, Node: CRIS-Pic, Next: CRIS-Regs, Prev: CRIS-Chars, Up: CRIS-Syntax

9.8.4.2 Symbols in position-independent code

.....

When generating position-independent code (SVR4 PIC) for use in `cris-axis-linux-gnu` or `crisv32-axis-linux-gnu` shared libraries, symbol suffixes are used to specify what kind of run-time symbol lookup will be used, expressed in the object as different `_relocation types_`. Usually,

all absolute symbol values must be located in a table, the `_global_offset_table_`, leaving the code position-independent; independent of values of global symbols and independent of the address of the code. The suffix modifies the value of the symbol, into for example an index into the global offset table where the real symbol value is entered, or a PC-relative value, or a value relative to the start of the global offset table. All symbol suffixes start with the character ':' (omitted in the list below). Every symbol use in code or a read-only section must therefore have a PIC suffix to enable a useful shared library to be created. Usually, these constructs must not be used with an additive constant offset as is usually allowed, i.e. no 4 as in 'symbol + 4' is allowed. This restriction is checked at link-time, not at assembly-time.

'GOT'

Attaching this suffix to a symbol in an instruction causes the symbol to be entered into the global offset table. The value is a 32-bit index for that symbol into the global offset table. The name of the corresponding relocation is 'R_CRIS_32_GOT'. Example: `'move.d [$r0+extsym:GOT],$r9'`

'GOT16'

Same as for 'GOT', but the value is a 16-bit index into the global offset table. The corresponding relocation is 'R_CRIS_16_GOT'. Example: `'move.d [$r0+asymbol:GOT16],$r10'`

'PLT'

This suffix is used for function symbols. It causes a `_procedure_linkage_table_`, an array of code stubs, to be created at the time the shared object is created or linked against, together with a global offset table entry. The value is a pc-relative offset to the corresponding stub code in the procedure linkage table. This arrangement causes the run-time symbol resolver to be called to look up and set the value of the symbol the first time the function is called (at latest; depending environment variables). It is only safe to leave the symbol unresolved this way if all references are function calls. The name of the relocation is 'R_CRIS_32_PLT_PCREL'. Example: `'add.d ffname:PLT,$pc'`

'PLTG'

Like PLT, but the value is relative to the beginning of the global offset table. The relocation is 'R_CRIS_32_PLT_GOTREL'. Example: `'move.d ffname:PLTG,$r3'`

'GOTPLT'

Similar to 'PLT', but the value of the symbol is a 32-bit index into the global offset table. This is somewhat of a mix between the effect of the 'GOT' and the 'PLT' suffix; the difference to 'GOT' is that there will be a procedure linkage table entry created, and that the symbol is assumed to be a function entry and will be resolved by the run-time resolver as with 'PLT'. The relocation is 'R_CRIS_32_GOTPLT'. Example: `'jsr [$r0+ffname:GOTPLT]'`

'GOTPLT16'

A variant of 'GOTPLT' giving a 16-bit value. Its relocation name is 'R_CRIS_16_GOTPLT'. Example: 'jsr [\$r0+fname:GOTPLT16]'

'GOTOFF'

This suffix must only be attached to a local symbol, but may be used in an expression adding an offset. The value is the address of the symbol relative to the start of the global offset table. The relocation name is 'R_CRIS_32_GOTREL'. Example: 'move.d [\$r0+localsym:GOTOFF],r3'



File: as.info, Node: CRIS-Regs, Next: CRIS-Pseudos, Prev: CRIS-Pic, Up: CRIS-Syntax

9.8.4.3 Register names

.....

A '\$' character may always prefix a general or special register name in an instruction operand but is mandatory when the option '--no-underscore' is specified or when the '.syntax register_prefix' directive is in effect (*note crisnous:). Register names are case-insensitive.



File: as.info, Node: CRIS-Pseudos, Prev: CRIS-Regs, Up: CRIS-Syntax

9.8.4.4 Assembler Directives

.....

There are a few CRIS-specific pseudo-directives in addition to the generic ones. *Note Pseudo Ops::. Constants emitted by pseudo-directives are in little-endian order for CRIS. There is no support for floating-point-specific directives for CRIS.

'dword EXPRESSIONS'

The 'dword' directive is a synonym for '.int', expecting zero or more EXPRESSIONS, separated by commas. For each expression, a 32-bit little-endian constant is emitted.

'syntax ARGUMENT'

The 'syntax' directive takes as ARGUMENT one of the following case-sensitive choices.

'no_register_prefix'

The 'syntax no_register_prefix' directive makes a '\$' character prefix on all registers optional. It overrides a previous setting, including the corresponding effect of the option '--no-underscore'. If this directive is used when ordinary symbols do not have a '_' character prefix, care must be taken to avoid ambiguities whether an operand is a register or a symbol; using symbols with names the same as general or special registers then invoke undefined behavior.

'register_prefix'

This directive makes a '\$' character prefix on all registers mandatory. It overrides a previous setting, including the corresponding effect of the option '--underscore'.

'leading_underscore'

This is an assertion directive, emitting an error if the '--no-underscore' option is in effect.

'no_leading_underscore'

This is the opposite of the '.syntax leading_underscore' directive and emits an error if the option '--underscore' is in effect.

'.arch ARGUMENT'

This is an assertion directive, giving an error if the specified ARGUMENT is not the same as the specified or default value for the '--march=ARCHITECTURE' option (*note march-option:).



File: as.info, Node: D10V-Dependent, Next: D30V-Dependent, Prev: CRIS-Dependent, Up: Machine Dependencies

9.9 D10V Dependent Features

=====

* Menu:

* D10V-Opts::	D10V Options
* D10V-Syntax::	Syntax
* D10V-Float::	Floating Point
* D10V-Opcodes::	Opcodes



File: as.info, Node: D10V-Opts, Next: D10V-Syntax, Up: D10V-Dependent

9.9.1 D10V Options

The Mitsubishi D10V version of 'as' has a few machine dependent options.

'-0'

The D10V can often execute two sub-instructions in parallel. When this option is used, 'as' will attempt to optimize its output by detecting when instructions can be executed in parallel.

'--nowarnswap'

To optimize execution performance, 'as' will sometimes swap the order of instructions. Normally this generates a warning. When this option is used, no warning will be generated when instructions are swapped.

'--gstabs-packing'

'--no-gstabs-packing'

'as' packs adjacent short instructions into a single packed instruction. '--no-gstabs-packing' turns instruction packing off if '--gstabs' is specified as well; '--gstabs-packing' (the default) turns instruction packing on even when '--gstabs' is specified.



File: as.info, Node: D10V-Syntax, Next: D10V-Float, Prev: D10V-Opts, Up: D10V-Dependent

9.9.2 Syntax

The D10V syntax is based on the syntax in Mitsubishi's D10V architecture manual. The differences are detailed below.

* Menu:

* D10V-Size::	Size Modifiers
* D10V-Subs::	Sub-Instructions
* D10V-Chars::	Special Characters
* D10V-Regs::	Register Names
* D10V-Addressing::	Addressing Modes
* D10V-Word::	@WORD Modifier



File: as.info, Node: D10V-Size, Next: D10V-Subs, Up: D10V-Syntax

9.9.2.1 Size Modifiers

.....

The D10V version of 'as' uses the instruction names in the D10V Architecture Manual. However, the names in the manual are sometimes ambiguous. There are instruction names that can assemble to a short or long form opcode. How does the assembler pick the correct form? 'as' will always pick the smallest form if it can. When dealing with a symbol that is not defined yet when a line is being assembled, it will always use the long form. If you need to force the assembler to use either the short or long form of the instruction, you can append either '.s' (short) or '.l' (long) to it. For example, if you are writing an assembly program and you want to do a branch to a symbol that is defined later in your program, you can write 'bra.s foo'. Objdump and GDB will always append '.s' or '.l' to instructions which have both short and long forms.



File: as.info, Node: D10V-Subs, Next: D10V-Chars, Prev: D10V-Size, Up: D10V-Syntax

9.9.2.2 Sub-Instructions

.....

The D10V assembler takes as input a series of instructions, either one-per-line, or in the special two-per-line format described in the next section. Some of these instructions will be short-form or sub-instructions. These sub-instructions can be packed into a single instruction. The assembler will do this automatically. It will also detect when it should not pack instructions. For example, when a label is defined, the next instruction will never be packaged with the previous one. Whenever a branch and link instruction is called, it will not be packaged with the next instruction so the return address will be valid. Nops are automatically inserted when necessary.

If you do not want the assembler automatically making these

decisions, you can control the packaging and execution type (parallel or sequential) with the special execution symbols described in the next section.

US

File: as.info, Node: D10V-Chars, Next: D10V-Regs, Prev: D10V-Subs, Up: D10V-Syntax

9.9.2.3 Special Characters

.....

A semicolon (';') can be used anywhere on a line to start a comment that extends to the end of the line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

Sub-instructions may be executed in order, in reverse-order, or in parallel. Instructions listed in the standard one-per-line format will be executed sequentially. To specify the executing order, use the following symbols:

'->'
Sequential with instruction on the left first.

'<-'
Sequential with instruction on the right first.

'||'
Parallel

The D10V syntax allows either one instruction per line, one instruction per line with the execution symbol, or two instructions per line. For example

'abs a1 -> abs r0'
Execute these sequentially. The instruction on the right is in the right container and is executed second.

'abs r0 <- abs a1'
Execute these reverse-sequentially. The instruction on the right is in the right container, and is executed first.

'ld2w r2,@r8+ || mac a0,r0,r7'
Execute these in parallel.

'ld2w r2,@r8+ ||'
'mac a0,r0,r7'
Two-line format. Execute these in parallel.

'ld2w r2,@r8+'
'mac a0,r0,r7'
Two-line format. Execute these sequentially. Assembler will put them in the proper containers.

'ld2w r2,@r8+ ->'
'mac a0,r0,r7'
Two-line format. Execute these sequentially. Same as above but second instruction will always go into right container.

Since '\$' has no special meaning, you may use it in symbol names.

US

File: as.info, Node: D10V-Regs, Next: D10V-Addressing, Prev: D10V-Chars, Up: D10V-Syntax

9.9.2.4 Register Names

.....

You can use the predefined symbols 'r0' through 'r15' to refer to the D10V registers. You can also use 'sp' as an alias for 'r15'. The accumulators are 'a0' and 'a1'. There are special register-pair names that may optionally be used in opcodes that require even-numbered registers. Register names are not case sensitive.

Register Pairs

```
'r0-r1'
'r2-r3'
'r4-r5'
'r6-r7'
'r8-r9'
'r10-r11'
'r12-r13'
'r14-r15'
```

The D10V also has predefined symbols for these control registers and status bits:

```
'psw'
    Processor Status Word
'bpsw'
    Backup Processor Status Word
'pc'
    Program Counter
'bpc'
    Backup Program Counter
'rpt_c'
    Repeat Count
'rpt_s'
    Repeat Start address
'rpt_e'
    Repeat End address
'mod_s'
    Modulo Start address
'mod_e'
    Modulo End address
'iba'
    Instruction Break Address
'f0'
    Flag 0
'f1'
    Flag 1
'c'
    Carry flag
```



File: as.info, Node: D10V-Addressing, Next: D10V-Word, Prev: D10V-Regs, Up: D10V-Syntax

9.9.2.5 Addressing Modes

.....

'as' understands the following addressing modes for the D10V. 'RN' in the following refers to any of the numbered registers, but not the control registers.

```
'RN'
    Register direct
'@RN'
```

```

    Register indirect
'@RN+'
    Register indirect with post-increment
'@RN-'
    Register indirect with post-decrement
'@-SP'
    Register indirect with pre-decrement
'@(DISP, RN)'
    Register indirect with displacement
'ADDR'
    PC relative address (for branch or rep).
'#IMM'
    Immediate data (the '#' is optional and ignored)

```



File: as.info, Node: D10V-Word, Prev: D10V-Addressing, Up: D10V-Syntax

9.9.2.6 @WORD Modifier

.....

Any symbol followed by '@word' will be replaced by the symbol's value shifted right by 2. This is used in situations such as loading a register with the address of a function (or any other code fragment). For example, if you want to load a register with the location of the function 'main' then jump to that function, you could do it as follows:

```

    ldi    r2, main@word
    jmp   r2

```



File: as.info, Node: D10V-Float, Next: D10V-Opcodes, Prev: D10V-Syntax, Up: D10V-Dependent

9.9.3 Floating Point

The D10V has no hardware floating point, but the '.float' and '.double' directives generates IEEE floating-point numbers for compatibility with other development tools.



File: as.info, Node: D10V-Opcodes, Prev: D10V-Float, Up: D10V-Dependent

9.9.4 Opcodes

For detailed information on the D10V machine instruction set, see 'D10V Architecture: A VLIW Microprocessor for Multimedia Applications' (Mitsubishi Electric Corp.). 'as' implements all the standard D10V opcodes. The only changes are those described in the section on size modifiers



File: as.info, Node: D30V-Dependent, Next: Epiphany-Dependent, Prev: D10V-Dependent, Up: Machine Dependencies

9.10 D30V Dependent Features

=====

* Menu:

```
* D30V-Opts::          D30V Options
* D30V-Syntax::       Syntax
* D30V-Float::        Floating Point
* D30V-Opcodes::     Opcodes
```



File: as.info, Node: D30V-Opts, Next: D30V-Syntax, Up: D30V-Dependent

9.10.1 D30V Options

The Mitsubishi D30V version of 'as' has a few machine dependent options.

'-O'

The D30V can often execute two sub-instructions in parallel. When this option is used, 'as' will attempt to optimize its output by detecting when instructions can be executed in parallel.

'-n'

When this option is used, 'as' will issue a warning every time it adds a nop instruction.

'-N'

When this option is used, 'as' will issue a warning if it needs to insert a nop after a 32-bit multiply before a load or 16-bit multiply instruction.



File: as.info, Node: D30V-Syntax, Next: D30V-Float, Prev: D30V-Opts, Up: D30V-Dependent

9.10.2 Syntax

The D30V syntax is based on the syntax in Mitsubishi's D30V architecture manual. The differences are detailed below.

* Menu:

```
* D30V-Size::          Size Modifiers
* D30V-Subs::         Sub-Instructions
* D30V-Chars::        Special Characters
* D30V-Guarded::      Guarded Execution
* D30V-Regs::         Register Names
* D30V-Addressing::   Addressing Modes
```



File: as.info, Node: D30V-Size, Next: D30V-Subs, Up: D30V-Syntax

9.10.2.1 Size Modifiers

.....

The D30V version of 'as' uses the instruction names in the D30V Architecture Manual. However, the names in the manual are sometimes ambiguous. There are instruction names that can assemble to a short or long form opcode. How does the assembler pick the correct form? 'as' will always pick the smallest form if it can. When dealing with a symbol that is not defined yet when a line is being assembled, it will

always use the long form. If you need to force the assembler to use either the short or long form of the instruction, you can append either '.s' (short) or '.l' (long) to it. For example, if you are writing an assembly program and you want to do a branch to a symbol that is defined later in your program, you can write 'bra.s foo'. Objdump and GDB will always append '.s' or '.l' to instructions which have both short and long forms.



File: as.info, Node: D30V-Subs, Next: D30V-Chars, Prev: D30V-Size, Up: D30V-Syntax

9.10.2.2 Sub-Instructions

.....

The D30V assembler takes as input a series of instructions, either one-per-line, or in the special two-per-line format described in the next section. Some of these instructions will be short-form or sub-instructions. These sub-instructions can be packed into a single instruction. The assembler will do this automatically. It will also detect when it should not pack instructions. For example, when a label is defined, the next instruction will never be packaged with the previous one. Whenever a branch and link instruction is called, it will not be packaged with the next instruction so the return address will be valid. Nops are automatically inserted when necessary.

If you do not want the assembler automatically making these decisions, you can control the packaging and execution type (parallel or sequential) with the special execution symbols described in the next section.



File: as.info, Node: D30V-Chars, Next: D30V-Guarded, Prev: D30V-Subs, Up: D30V-Syntax

9.10.2.3 Special Characters

.....

A semicolon (';') can be used anywhere on a line to start a comment that extends to the end of the line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

Sub-instructions may be executed in order, in reverse-order, or in parallel. Instructions listed in the standard one-per-line format will be executed sequentially unless you use the '-0' option.

To specify the executing order, use the following symbols:

'->'

Sequential with instruction on the left first.

Sequential with instruction on the right first.

'||'

Parallel

The D30V syntax allows either one instruction per line, one instruction per line with the execution symbol, or two instructions per line. For example

```
'abs r2,r3 -> abs r4,r5'
```

Execute these sequentially. The instruction on the right is in the right container and is executed second.

```
'abs r2,r3 <- abs r4,r5'
```

Execute these reverse-sequentially. The instruction on the right is in the right container, and is executed first.

```
'abs r2,r3 || abs r4,r5'
```

Execute these in parallel.

```
'ldw r2,@(r3,r4) ||'
```

```
'mulx r6,r8,r9'
```

Two-line format. Execute these in parallel.

```
'mulx a0,r8,r9'
```

```
'stw r2,@(r3,r4)'
```

Two-line format. Execute these sequentially unless '-0' option is used. If the '-0' option is used, the assembler will determine if the instructions could be done in parallel (the above two instructions can be done in parallel), and if so, emit them as parallel instructions. The assembler will put them in the proper containers. In the above example, the assembler will put the 'stw' instruction in left container and the 'mulx' instruction in the right container.

```
'stw r2,@(r3,r4) ->'
```

```
'mulx a0,r8,r9'
```

Two-line format. Execute the 'stw' instruction followed by the 'mulx' instruction sequentially. The first instruction goes in the left container and the second instruction goes into right container. The assembler will give an error if the machine ordering constraints are violated.

```
'stw r2,@(r3,r4) <-'
```

```
'mulx a0,r8,r9'
```

Same as previous example, except that the 'mulx' instruction is executed before the 'stw' instruction.

Since '\$' has no special meaning, you may use it in symbol names.



File: as.info, Node: D30V-Guarded, Next: D30V-Regs, Prev: D30V-Chars, Up: D30V-Syntax

9.10.2.4 Guarded Execution

.....

'as' supports the full range of guarded execution directives for each instruction. Just append the directive after the instruction proper. The directives are:

```
'/tx'
```

Execute the instruction if flag f0 is true.

```
'/fx'
```

```

Execute the instruction if flag f0 is false.
'/xt'
Execute the instruction if flag f1 is true.
'/xf'
Execute the instruction if flag f1 is false.
'/tt'
Execute the instruction if both flags f0 and f1 are true.
'/tf'
Execute the instruction if flag f0 is true and flag f1 is false.

```



File: as.info, Node: D30V-Regs, Next: D30V-Addressing, Prev: D30V-Guarded, Up: D30V-Syntax

9.10.2.5 Register Names

.....

You can use the predefined symbols 'r0' through 'r63' to refer to the D30V registers. You can also use 'sp' as an alias for 'r63' and 'link' as an alias for 'r62'. The accumulators are 'a0' and 'a1'.

The D30V also has predefined symbols for these control registers and status bits:

```

'psw'
Processor Status Word
'bpsw'
Backup Processor Status Word
'pc'
Program Counter
'bpc'
Backup Program Counter
'rpt_c'
Repeat Count
'rpt_s'
Repeat Start address
'rpt_e'
Repeat End address
'mod_s'
Modulo Start address
'mod_e'
Modulo End address
'iba'
Instruction Break Address
'f0'
Flag 0
'f1'
Flag 1
'f2'
Flag 2
'f3'
Flag 3
'f4'
Flag 4
'f5'
Flag 5
'f6'
Flag 6
'f7'
Flag 7

```

```
's'      Same as flag 4 (saturation flag)
'v'      Same as flag 5 (overflow flag)
'va'     Same as flag 6 (sticky overflow flag)
'c'      Same as flag 7 (carry/borrow flag)
'b'      Same as flag 7 (carry/borrow flag)
```



File: as.info, Node: D30V-Addressing, Prev: D30V-Regs, Up: D30V-Syntax

9.10.2.6 Addressing Modes

.....

'as' understands the following addressing modes for the D30V. 'RN' in the following refers to any of the numbered registers, but not the control registers.

```
'RN'      Register direct
'@RN'     Register indirect
'@RN+'    Register indirect with post-increment
'@RN-'    Register indirect with post-decrement
'@-SP'    Register indirect with pre-decrement
'@(DISP, RN)' Register indirect with displacement
'ADDR'    PC relative address (for branch or rep).
'#IMM'    Immediate data (the '#' is optional and ignored)
```



File: as.info, Node: D30V-Float, Next: D30V-Opcodes, Prev: D30V-Syntax, Up: D30V-Dependent

9.10.3 Floating Point

The D30V has no hardware floating point, but the '.float' and '.double' directives generates IEEE floating-point numbers for compatibility with other development tools.



File: as.info, Node: D30V-Opcodes, Prev: D30V-Float, Up: D30V-Dependent

9.10.4 Opcodes

For detailed information on the D30V machine instruction set, see 'D30V Architecture: A VLIW Microprocessor for Multimedia Applications' (Mitsubishi Electric Corp.). 'as' implements all the standard D30V opcodes. The only changes are those described in the section on size modifiers



File: as.info, Node: Epiphany-Dependent, Next: H8/300-Dependent, Prev: D30V-Dependent, Up: Machine Dependencies

9.11 Epiphany Dependent Features

=====

* Menu:

* Epiphany Options:: Options
 * Epiphany Syntax:: Epiphany Syntax



File: as.info, Node: Epiphany Options, Next: Epiphany Syntax, Up: Epiphany-Dependent

9.11.1 Options

'as' has two additional command-line options for the Epiphany architecture.

'-mepiphany'
 Specifies that the both 32 and 16 bit instructions are allowed.
 This is the default behavior.

'-mepiphany16'
 Restricts the permitted instructions to just the 16 bit set.



File: as.info, Node: Epiphany Syntax, Prev: Epiphany Options, Up: Epiphany-Dependent

9.11.2 Epiphany Syntax

* Menu:

* Epiphany-Chars:: Special Characters



File: as.info, Node: Epiphany-Chars, Up: Epiphany Syntax

9.11.2.1 Special Characters

.....

The presence of a ';' on a line indicates the start of a comment that extends to the end of the current line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

The '`' character can be used to separate statements on the same line.



File: as.info, Node: H8/300-Dependent, Next: HPPA-Dependent, Prev: Epiphany-Dependent, Up: Machine Dependencies

9.12 H8/300 Dependent Features

=====

* Menu:

* H8/300 Options::	Options
* H8/300 Syntax::	Syntax
* H8/300 Floating Point::	Floating Point
* H8/300 Directives::	H8/300 Machine Directives
* H8/300 Opcodes::	Opcodes



File: as.info, Node: H8/300 Options, Next: H8/300 Syntax, Up: H8/300-Dependent

9.12.1 Options

The Renesas H8/300 version of 'as' has one machine-dependent option:

'-h-tick-hex'

Support H'00 style hex constants in addition to 0x00 style.

'-mach=NAME'

Sets the H8300 machine variant. The following machine names are recognised: 'h8300h', 'h8300hn', 'h8300s', 'h8300sn', 'h8300sx' and 'h8300sxn'.



File: as.info, Node: H8/300 Syntax, Next: H8/300 Floating Point, Prev: H8/300 Options, Up: H8/300-Dependent

9.12.2 Syntax

* Menu:

* H8/300-Chars::	Special Characters
* H8/300-Regs::	Register Names
* H8/300-Addressing::	Addressing Modes



File: as.info, Node: H8/300-Chars, Next: H8/300-Regs, Up: H8/300 Syntax

9.12.2.1 Special Characters

.....

';' is the line comment character.

'\$' can be used instead of a newline to separate statements. Therefore you may not use '\$' in symbol names on the H8/300.



File: as.info, Node: H8/300-Regs, Next: H8/300-Addressing, Prev: H8/300-Chars, Up: H8/300 Syntax

9.12.2.2 Register Names

.....

You can use predefined symbols of the form 'rNh' and 'rNl' to refer to the H8/300 registers as sixteen 8-bit general-purpose registers. N is a digit from '0' to '7'); for instance, both 'r0h' and 'r7l' are valid register names.

You can also use the eight predefined symbols 'rN' to refer to the H8/300 registers as 16-bit registers (you must use this form for addressing).

On the H8/300H, you can also use the eight predefined symbols 'erN' ('er0' ... 'er7') to refer to the 32-bit general purpose registers.

The two control registers are called 'pc' (program counter; a 16-bit register, except on the H8/300H where it is 24 bits) and 'ccr' (condition code register; an 8-bit register). 'r7' is used as the stack pointer, and can also be called 'sp'.



File: as.info, Node: H8/300-Addressing, Prev: H8/300-Regs, Up: H8/300 Syntax

9.12.2.3 Addressing Modes

.....

as understands the following addressing modes for the H8/300:

- 'rN'
Register direct
- '@rN'
Register indirect
- '@(D, rN)'
'@(D:16, rN)'
'@(D:24, rN)'
Register indirect: 16-bit or 24-bit displacement D from register N.
(24-bit displacements are only meaningful on the H8/300H.)
- '@rN+'
Register indirect with post-increment
- '@-rN'
Register indirect with pre-decrement
- '@AA'
'@AA:8'
'@AA:16'
'@AA:24'
Absolute address 'aa'. (The address size ':24' only makes sense on the H8/300H.)
- '#XX'
'#XX:8'
'#XX:16'
'#XX:32'
Immediate data XX. You may specify the ':8', ':16', or ':32' for clarity, if you wish; but 'as' neither requires this nor uses it--the data size required is taken from context.

'@@AA'

'@@AA:8'

Memory indirect. You may specify the ':8' for clarity, if you wish; but 'as' neither requires this nor uses it.

US

File: as.info, Node: H8/300 Floating Point, Next: H8/300 Directives, Prev: H8/300 Syntax, Up: H8/300-Dependent

9.12.3 Floating Point

The H8/300 family has no hardware floating point, but the '.float' directive generates IEEE floating-point numbers for compatibility with other development tools.

US

File: as.info, Node: H8/300 Directives, Next: H8/300 Opcodes, Prev: H8/300 Floating Point, Up: H8/300-Dependent

9.12.4 H8/300 Machine Directives

'as' has the following machine-dependent directives for the H8/300:

'.h8300h'

Recognize and emit additional instructions for the H8/300H variant, and also make '.int' emit 32-bit numbers rather than the usual (16-bit) for the H8/300 family.

'.h8300s'

Recognize and emit additional instructions for the H8S variant, and also make '.int' emit 32-bit numbers rather than the usual (16-bit) for the H8/300 family.

'.h8300hn'

Recognize and emit additional instructions for the H8/300H variant in normal mode, and also make '.int' emit 32-bit numbers rather than the usual (16-bit) for the H8/300 family.

'.h8300sn'

Recognize and emit additional instructions for the H8S variant in normal mode, and also make '.int' emit 32-bit numbers rather than the usual (16-bit) for the H8/300 family.

On the H8/300 family (including the H8/300H) '.word' directives generate 16-bit numbers.

US

File: as.info, Node: H8/300 Opcodes, Prev: H8/300 Directives, Up: H8/300-Dependent

9.12.5 Opcodes

For detailed information on the H8/300 machine instruction set, see 'H8/300 Series Programming Manual'. For information specific to the H8/300H, see 'H8/300H Series Programming Manual' (Renesas).

'as' implements all the standard H8/300 opcodes. No additional

pseudo-instructions are needed on this family.

The following table summarizes the H8/300 opcodes, and their arguments. Entries marked '*' are opcodes used only on the H8/300H.

Legend:

Rs source register
 Rd destination register
 abs absolute address
 imm immediate data
 disp:N N-bit displacement from a register
 pcrel:N N-bit displacement relative to program counter

add.b #imm,rd	* andc #imm,ccr
add.b rs,rd	band #imm,rd
add.w rs,rd	band #imm,@rd
* add.w #imm,rd	band #imm,@abs:8
* add.l rs,rd	bra pcrel:8
* add.l #imm,rd	* bra pcrel:16
adds #imm,rd	bt pcrel:8
addx #imm,rd	* bt pcrel:16
addx rs,rd	brn pcrel:8
and.b #imm,rd	* brn pcrel:16
and.b rs,rd	bf pcrel:8
* and.w rs,rd	* bf pcrel:16
* and.w #imm,rd	bhi pcrel:8
* and.l #imm,rd	* bhi pcrel:16
* and.l rs,rd	bls pcrel:8
* bls pcrel:16	bld #imm,rd
bcc pcrel:8	bld #imm,@rd
* bcc pcrel:16	bld #imm,@abs:8
bhs pcrel:8	bnot #imm,rd
* bhs pcrel:16	bnot #imm,@rd
bcs pcrel:8	bnot #imm,@abs:8
* bcs pcrel:16	bnot rs,rd
blo pcrel:8	bnot rs,@rd
* blo pcrel:16	bnot rs,@abs:8
bne pcrel:8	bor #imm,rd
* bne pcrel:16	bor #imm,@rd
beq pcrel:8	bor #imm,@abs:8
* beq pcrel:16	bset #imm,rd
bvc pcrel:8	bset #imm,@rd
* bvc pcrel:16	bset #imm,@abs:8
bvs pcrel:8	bset rs,rd
* bvs pcrel:16	bset rs,@rd
bpl pcrel:8	bset rs,@abs:8
* bpl pcrel:16	bsr pcrel:8
bmi pcrel:8	bsr pcrel:16
* bmi pcrel:16	bst #imm,rd
bge pcrel:8	bst #imm,@rd
* bge pcrel:16	bst #imm,@abs:8
blt pcrel:8	btst #imm,rd
* blt pcrel:16	btst #imm,@rd
bgt pcrel:8	btst #imm,@abs:8
* bgt pcrel:16	btst rs,rd
ble pcrel:8	btst rs,@rd
* ble pcrel:16	btst rs,@abs:8
bclr #imm,rd	bxor #imm,rd
bclr #imm,@rd	bxor #imm,@rd

bclr #imm,@abs:8	bxor #imm,@abs:8
bclr rs,rd	cmp.b #imm,rd
bclr rs,@rd	cmp.b rs,rd
bclr rs,@abs:8	cmp.w rs,rd
biand #imm,rd	cmp.w rs,rd
biand #imm,@rd	* cmp.w #imm,rd
biand #imm,@abs:8	* cmp.l #imm,rd
bild #imm,rd	* cmp.l rs,rd
bild #imm,@rd	daa rs
bild #imm,@abs:8	das rs
bior #imm,rd	dec.b rs
bior #imm,@rd	* dec.w #imm,rd
bior #imm,@abs:8	* dec.l #imm,rd
bist #imm,rd	divxu.b rs,rd
bist #imm,@rd	* divxu.w rs,rd
bist #imm,@abs:8	* divxs.b rs,rd
bixor #imm,rd	* divxs.w rs,rd
bixor #imm,@rd	eepmov
bixor #imm,@abs:8	* eepmovw
* exts.w rd	mov.w rs,@abs:16
* exts.l rd	* mov.l #imm,rd
* extu.w rd	* mov.l rs,rd
* extu.l rd	* mov.l @rs,rd
inc rs	* mov.l @(disp:16,rs),rd
* inc.w #imm,rd	* mov.l @(disp:24,rs),rd
* inc.l #imm,rd	* mov.l @rs+,rd
jmp @rs	* mov.l @abs:16,rd
jmp abs	* mov.l @abs:24,rd
jmp @@abs:8	* mov.l rs,@rd
jsr @rs	* mov.l rs,@(disp:16,rd)
jsr abs	* mov.l rs,@(disp:24,rd)
jsr @@abs:8	* mov.l rs,@-rd
ldc #imm,ccr	* mov.l rs,@abs:16
ldc rs,ccr	* mov.l rs,@abs:24
* ldc @abs:16,ccr	movfpe @abs:16,rd
* ldc @abs:24,ccr	movtpe rs,@abs:16
* ldc @(disp:16,rs),ccr	mulxu.b rs,rd
* ldc @(disp:24,rs),ccr	* mulxu.w rs,rd
* ldc @rs+,ccr	* mulxs.b rs,rd
* ldc @rs,ccr	* mulxs.w rs,rd
* mov.b @(disp:24,rs),rd	neg.b rs
* mov.b rs,@(disp:24,rd)	* neg.w rs
mov.b @abs:16,rd	* neg.l rs
mov.b rs,rd	nop
mov.b @abs:8,rd	not.b rs
mov.b rs,@abs:8	* not.w rs
mov.b rs,rd	* not.l rs
mov.b #imm,rd	or.b #imm,rd
mov.b @rs,rd	or.b rs,rd
mov.b @(disp:16,rs),rd	* or.w #imm,rd
mov.b @rs+,rd	* or.w rs,rd
mov.b @abs:8,rd	* or.l #imm,rd
mov.b rs,@rd	* or.l rs,rd
mov.b rs,@(disp:16,rd)	orc #imm,ccr
mov.b rs,@-rd	pop.w rs
mov.b rs,@abs:8	* pop.l rs
mov.w rs,@rd	push.w rs
* mov.w @(disp:24,rs),rd	* push.l rs
* mov.w rs,@(disp:24,rd)	rotl.b rs

```

* mov.w @abs:24,rd
* mov.w rs,@abs:24
  mov.w rs,rd
  mov.w #imm,rd
  mov.w @rs,rd
  mov.w @(disp:16,rs),rd
  mov.w @rs+,rd
  mov.w @abs:16,rd
  mov.w rs,@(disp:16,rd)
  mov.w rs,@-rd
* rotxr.l rs
  bpt
  rte
  rts
  shal.b rs
* shal.w rs
* shal.l rs
  shar.b rs
* shar.w rs
* shar.l rs
  shll.b rs
* shll.w rs
* shll.l rs
  shlr.b rs
* shlr.w rs
* shlr.l rs
  sleep
  stc ccr,rd
* stc ccr,@rs
* stc ccr,@(disp:16,rd)
* rotl.w rs
* rotl.l rs
  rotr.b rs
* rotr.w rs
* rotr.l rs
  rotxl.b rs
* rotxl.w rs
* rotxl.l rs
  rotxr.b rs
* rotxr.w rs
* stc ccr,@(disp:24,rd)
* stc ccr,@-rd
* stc ccr,@abs:16
* stc ccr,@abs:24
  sub.b rs,rd
  sub.w rs,rd
* sub.w #imm,rd
* sub.l rs,rd
* sub.l #imm,rd
  subs #imm,rd
  subx #imm,rd
  subx rs,rd
* trapa #imm
  xor #imm,rd
  xor rs,rd
* xor.w #imm,rd
* xor.w rs,rd
* xor.l #imm,rd
* xor.l rs,rd
  xorc #imm,ccr

```

Four H8/300 instructions ('add', 'cmp', 'mov', 'sub') are defined with variants using the suffixes '.b', '.w', and '.l' to specify the size of a memory operand. 'as' supports these suffixes, but does not require them; since one of the operands is always a register, 'as' can deduce the correct size.

For example, since 'r0' refers to a 16-bit register,

```
mov    r0,@foo
```

is equivalent to

```
mov.w  r0,@foo
```

If you use the size suffixes, 'as' issues a warning when the suffix and the register size do not match.



File: as.info, Node: HPPA-Dependent, Next: ESA/390-Dependent, Prev: H8/300-Dependent, Up: Machine Dependencies

9.13 HPPA Dependent Features

=====

* Menu:

```

* HPPA Notes::          Notes
* HPPA Options::       Options
* HPPA Syntax::        Syntax
* HPPA Floating Point:: Floating Point
* HPPA Directives::    HPPA Machine Directives

```

* HPPA Opcodes:: Opcodes



File: as.info, Node: HPPA Notes, Next: HPPA Options, Up: HPPA-Dependent

9.13.1 Notes

As a back end for GNU CC 'as' has been thoroughly tested and should work extremely well. We have tested it only minimally on hand written assembly code and no one has tested it much on the assembly output from the HP compilers.

The format of the debugging sections has changed since the original 'as' port (version 1.3X) was released; therefore, you must rebuild all HPPA objects and libraries with the new assembler so that you can debug the final executable.

The HPPA 'as' port generates a small subset of the relocations available in the SOM and ELF object file formats. Additional relocation support will be added as it becomes necessary.



File: as.info, Node: HPPA Options, Next: HPPA Syntax, Prev: HPPA Notes, Up: HPPA-Dependent

9.13.2 Options

'as' has no machine-dependent command-line options for the HPPA.



File: as.info, Node: HPPA Syntax, Next: HPPA Floating Point, Prev: HPPA Options, Up: HPPA-Dependent

9.13.3 Syntax

The assembler syntax closely follows the HPPA instruction set reference manual; assembler directives and general syntax closely follow the HPPA assembly language reference manual, with a few noteworthy differences.

First, a colon may immediately follow a label definition. This is simply for compatibility with how most assembly language programmers write code.

Some obscure expression parsing problems may affect hand written code which uses the 'spop' instructions, or code which makes significant use of the '!' line separator.

'as' is much less forgiving about missing arguments and other similar oversights than the HP assembler. 'as' notifies you of missing arguments as syntax errors; this is regarded as a feature, not a bug.

Finally, 'as' allows you to use an external symbol without explicitly importing the symbol. `_Warning:_` in the future this will be an error for HPPA targets.

Special characters for HPPA targets include:

';' is the line comment character.

'!' can be used instead of a newline to separate statements.

Since '\$' has no special meaning, you may use it in symbol names.



File: as.info, Node: HPPA Floating Point, Next: HPPA Directives, Prev: HPPA Syntax, Up: HPPA-Dependent

9.13.4 Floating Point

The HPPA family uses IEEE floating-point numbers.



File: as.info, Node: HPPA Directives, Next: HPPA Opcodes, Prev: HPPA Floating Point, Up: HPPA-Dependent

9.13.5 HPPA Assembler Directives

'as' for the HPPA supports many additional directives for compatibility with the native assembler. This section describes them only briefly. For detailed information on HPPA-specific assembler directives, see 'HP9000 Series 800 Assembly Language Reference Manual' (HP 92432-90001).

'as' does not support the following assembler directives described in the HP manual:

```
.endm          .liston
.enter         .locct
.leave        .macro
.listoff
```

Beyond those implemented for compatibility, 'as' supports one additional assembler directive for the HPPA: '.param'. It conveys register argument locations for static functions. Its syntax closely follows the '.export' directive.

These are the additional directives in 'as' for the HPPA:

'`.block N`'

'`.blockz N`'

Reserve N bytes of storage, and initialize them to zero.

'`.call`'

Mark the beginning of a procedure call. Only the special case with `_no arguments_` is allowed.

'`.callinfo [PARAM=VALUE, ...] [FLAG, ...]`'

Specify a number of parameters and flags that define the environment for a procedure.

PARAM may be any of 'frame' (frame size), 'entry_gr' (end of general register range), 'entry_fr' (end of float register range), 'entry_sr' (end of space register range).

The values for FLAG are 'calls' or 'caller' (proc has subroutines), 'no_calls' (proc does not call subroutines), 'save_rp' (preserve return pointer), 'save_sp' (proc preserves stack pointer), 'no_unwind' (do not unwind this proc), 'hpux_int' (proc is interrupt routine).

`code`

Assemble into the standard section called '\$TEXT\$', subsection '\$CODE\$'.

`copyright "STRING"`

In the SOM object format, insert STRING into the object code, marked as a copyright string.

`copyright "STRING"`

In the ELF object format, insert STRING into the object code, marked as a version string.

`enter`

Not yet supported; the assembler rejects programs containing this directive.

`entry`

Mark the beginning of a procedure.

`exit`

Mark the end of a procedure.

`export NAME [,TYP] [,PARAM=R]`

Make a procedure NAME available to callers. TYP, if present, must be one of 'absolute', 'code' (ELF only, not SOM), 'data', 'entry', 'data', 'entry', 'millicode', 'plabel', 'pri_prog', or 'sec_prog'.

PARAM, if present, provides either relocation information for the procedure arguments and result, or a privilege level. PARAM may be 'argwN' (where N ranges from '0' to '3', and indicates one of four one-word arguments); 'rtnval' (the procedure's result); or 'priv_lev' (privilege level). For arguments or the result, R specifies how to relocate, and must be one of 'no' (not relocatable), 'gr' (argument is in general register), 'fr' (in floating point register), or 'fu' (upper half of float register). For 'priv_lev', R is an integer.

`half N`

Define a two-byte integer constant N; synonym for the portable 'as' directive 'short'.

`import NAME [,TYP]`

Converse of 'export'; make a procedure available to call. The arguments use the same conventions as the first two arguments for 'export'.

`label NAME`

Define NAME as a label for the current assembly location.

`leave`

Not yet supported; the assembler rejects programs containing this directive.

`'.origin LC'`

Advance location counter to LC. Synonym for the 'as' portable directive `'.org'`.

`'.param NAME [,TYP] [,PARAM=R]'`

Similar to `'.export'`, but used for static procedures.

`'.proc'`

Use preceding the first statement of a procedure.

`'.procend'`

Use following the last statement of a procedure.

`'LABEL .reg EXPR'`

Synonym for `'.equ'`; define LABEL with the absolute expression EXPR as its value.

`'.space SECNAME [,PARAMS]'`

Switch to section SECNAME, creating a new section by that name if necessary. You may only use PARAMS when creating a new section, not when switching to an existing one. SECNAME may identify a section by number rather than by name.

If specified, the list PARAMS declares attributes of the section, identified by keywords. The keywords recognized are `'spnum=EXP'` (identify this section by the number EXP, an absolute expression), `'sort=EXP'` (order sections according to this sort key when linking; EXP is an absolute expression), `'unloadable'` (section contains no loadable data), `'notdefined'` (this section defined elsewhere), and `'private'` (data in this section not available to other programs).

`'.spnum SECNAM'`

Allocate four bytes of storage, and initialize them with the section number of the section named SECNAM. (You can define the section number with the HPPA `'.space'` directive.)

`'.string "STR"'`

Copy the characters in the string STR to the object file. *Note Strings: Strings, for information on escape sequences you can use in 'as' strings.

Warning! The HPPA version of `'.string'` differs from the usual 'as' definition: it does not write a zero byte after copying STR.

`'.stringz "STR"'`

Like `'.string'`, but appends a zero byte after copying STR to object file.

`'.subspa NAME [,PARAMS]'`

`'.nsubspa NAME [,PARAMS]'`

Similar to `'.space'`, but selects a subsection NAME within the current section. You may only specify PARAMS when you create a subsection (in the first instance of `'.subspa'` for this NAME).

If specified, the list PARAMS declares attributes of the subsection, identified by keywords. The keywords recognized are `'quad=EXPR'` ("quadrant" for this subsection), `'align=EXPR'` (alignment for beginning of this subsection; a power of two), `'access=EXPR'` (value for "access rights" field), `'sort=EXPR'`

(sorting order for this subspace in link), 'code_only' (subsection contains only code), 'unloadable' (subsection cannot be loaded into memory), 'comdat' (subsection is comdat), 'common' (subsection is common block), 'dup_comm' (subsection may have duplicate names), or 'zero' (subsection is all zeros, do not write in object file).

'.nsubspa' always creates a new subspace with the given name, even if one with the same name already exists.

'comdat', 'common' and 'dup_comm' can be used to implement various flavors of one-only support when using the SOM linker. The SOM linker only supports specific combinations of these flags. The details are not documented. A brief description is provided here.

'comdat' provides a form of linkonce support. It is useful for both code and data subspaces. A 'comdat' subspace has a key symbol marked by the 'is_comdat' flag or 'ST_COMDAT'. Only the first subspace for any given key is selected. The key symbol becomes universal in shared links. This is similar to the behavior of 'secondary_def' symbols.

'common' provides Fortran named common support. It is only useful for data subspaces. Symbols with the flag 'is_common' retain this flag in shared links. Referencing a 'is_common' symbol in a shared library from outside the library doesn't work. Thus, 'is_common' symbols must be output whenever they are needed.

'common' and 'dup_comm' together provide Cobol common support. The subspaces in this case must all be the same length. Otherwise, this support is similar to the Fortran common support.

'dup_comm' by itself provides a type of one-only support for code. Only the first 'dup_comm' subspace is selected. There is a rather complex algorithm to compare subspaces. Code symbols marked with the 'dup_common' flag are hidden. This support was intended for "C++ duplicate inlines".

A simplified technique is used to mark the flags of symbols based on the flags of their subspace. A symbol with the scope SS_UNIVERSAL and type ST_ENTRY, ST_CODE or ST_DATA is marked with the corresponding settings of 'comdat', 'common' and 'dup_comm' from the subspace, respectively. This avoids having to introduce additional directives to mark these symbols. The HP assembler sets 'is_common' from 'common'. However, it doesn't set the 'dup_common' from 'dup_comm'. It doesn't have 'comdat' support.

'.version "STR"'

Write STR as version identifier in object code.



File: as.info, Node: HPPA Opcodes, Prev: HPPA Directives, Up: HPPA-Dependent

9.13.6 Opcodes

For detailed information on the HPPA machine instruction set, see 'PA-RISC Architecture and Instruction Set Reference Manual' (HP 09740-90039).



File: as.info, Node: ESA/390-Dependent, Next: i386-Dependent, Prev: HPPA-Dependent, Up: Machine Dependencies

9.14 ESA/390 Dependent Features

=====

* Menu:

* ESA/390 Notes::	Notes
* ESA/390 Options::	Options
* ESA/390 Syntax::	Syntax
* ESA/390 Floating Point::	Floating Point
* ESA/390 Directives::	ESA/390 Machine Directives
* ESA/390 Opcodes::	Opcodes



File: as.info, Node: ESA/390 Notes, Next: ESA/390 Options, Up: ESA/390-Dependent

9.14.1 Notes

The ESA/390 'as' port is currently intended to be a back-end for the GNU CC compiler. It is not HLASM compatible, although it does support a subset of some of the HLASM directives. The only supported binary file format is ELF; none of the usual MVS/VM/OE/USS object file formats, such as ESD or XSD, are supported.

When used with the GNU CC compiler, the ESA/390 'as' will produce correct, fully relocated, functional binaries, and has been used to compile and execute large projects. However, many aspects should still be considered experimental; these include shared library support, dynamically loadable objects, and any relocation other than the 31-bit relocation.



File: as.info, Node: ESA/390 Options, Next: ESA/390 Syntax, Prev: ESA/390 Notes, Up: ESA/390-Dependent

9.14.2 Options

'as' has no machine-dependent command-line options for the ESA/390.



File: as.info, Node: ESA/390 Syntax, Next: ESA/390 Floating Point, Prev: ESA/390 Options, Up: ESA/390-Dependent

9.14.3 Syntax

The opcode/operand syntax follows the ESA/390 Principles of Operation manual; assembler directives and general syntax are loosely based on the prevailing AT&T/SVR4/ELF/Solaris style notation. HLASM-style directives are not supported for the most part, with the exception of those described herein.

A leading dot in front of directives is optional, and the case of directives is ignored; thus for example, .using and USING have the same

effect.

A colon may immediately follow a label definition. This is simply for compatibility with how most assembly language programmers write code.

'#' is the line comment character.

';' can be used instead of a newline to separate statements.

Since '\$' has no special meaning, you may use it in symbol names.

Registers can be given the symbolic names r0..r15, fp0, fp2, fp4, fp6. By using these symbolic names, 'as' can detect simple syntax errors. The name rarg or r.arg is a synonym for r11, rtca or r.tca for r12, sp, r.sp, dsa r.dsa for r13, lr or r.lr for r14, rbase or r.base for r3 and rpgt or r.pgt for r4.

'*' is the current location counter. Unlike '.' it is always relative to the last USING directive. Note that this means that expressions cannot use multiplication, as any occurrence of '*' will be interpreted as a location counter.

All labels are relative to the last USING. Thus, branches to a label always imply the use of base+displacement.

Many of the usual forms of address constants / address literals are supported. Thus,

```
.using *,r3
L   r15,=A(some_routine)
LM  r6,r7,=V(some_longlong_extern)
A   r1,=F'12'
AH  r0,=H'42'
ME  r6,=E'3.1416'
MD  r6,=D'3.14159265358979'
O   r6,=XL4'cacad0d0'
.ltorg
```

should all behave as expected: that is, an entry in the literal pool will be created (or reused if it already exists), and the instruction operands will be the displacement into the literal pool using the current base register (as last declared with the '.using' directive).



File: as.info, Node: ESA/390 Floating Point, Next: ESA/390 Directives, Prev: ESA/390 Syntax, Up: ESA/390-Dependent

9.14.4 Floating Point

The assembler generates only IEEE floating-point numbers. The older floating point formats are not supported.



File: as.info, Node: ESA/390 Directives, Next: ESA/390 Opcodes, Prev: ESA/390 Floating Point, Up: ESA/390-Dependent

9.14.5 ESA/390 Assembler Directives

'as' for the ESA/390 supports all of the standard ELF/SVR4 assembler directives that are documented in the main part of this documentation. Several additional directives are supported in order to implement the ESA/390 addressing model. The most important of these are '.using' and '.ltorg'

These are the additional directives in 'as' for the ESA/390:

'.dc'

A small subset of the usual DC directive is supported.

'.drop REGNO'

Stop using REGNO as the base register. The REGNO must have been previously declared with a '.using' directive in the same section as the current section.

'.ebcdic STRING'

Emit the EBCDIC equivalent of the indicated string. The emitted string will be null terminated. Note that the directives '.string' etc. emit ascii strings by default.

'EQU'

The standard HLASM-style EQU directive is not supported; however, the standard 'as' directive .equ can be used to the same effect.

'.ltorg'

Dump the literal pool accumulated so far; begin a new literal pool. The literal pool will be written in the current section; in order to generate correct assembly, a '.using' must have been previously specified in the same section.

'.using EXPR,REGNO'

Use REGNO as the base register for all subsequent RX, RS, and SS form instructions. The EXPR will be evaluated to obtain the base address; usually, EXPR will merely be '*'.

This assembler allows two '.using' directives to be simultaneously outstanding, one in the '.text' section, and one in another section (typically, the '.data' section). This feature allows dynamically loaded objects to be implemented in a relatively straightforward way. A '.using' directive must always be specified in the '.text' section; this will specify the base register that will be used for branches in the '.text' section. A second '.using' may be specified in another section; this will specify the base register that is used for non-label address literals. When a second '.using' is specified, then the subsequent '.ltorg' must be put in the same section; otherwise an error will result.

Thus, for example, the following code uses 'r3' to address branch targets and 'r4' to address the literal pool, which has been written to the '.data' section. The is, the constants '=A(some_routine)', '=H'42'' and '=E'3.1416'' will all appear in the '.data' section.

```
.data
    .using  LITPOOL,r4
.text
    BASR   r3,0
    .using *,r3
```

```

        B      START
    .long  LITPOOL
START:
    L    r4,4(,r3)
    L    r15,=A(some_routine)
    LTR  r15,r15
    BNE LABEL
    AH  r0,=H'42'
LABEL:
    ME  r6,=E'3.1416'
.data
LITPOOL:
    .ltorg

```

Note that this dual-'.using' directive semantics extends and is not compatible with HLASM semantics. Note that this assembler directive does not support the full range of HLASM semantics.



File: as.info, Node: ESA/390 Opcodes, Prev: ESA/390 Directives, Up: ESA/390-Dependent

9.14.6 Opcodes

For detailed information on the ESA/390 machine instruction set, see 'ESA/390 Principles of Operation' (IBM Publication Number DZ9AR004).



File: as.info, Node: i386-Dependent, Next: i860-Dependent, Prev: ESA/390-Dependent, Up: Machine Dependencies

9.15 80386 Dependent Features

=====

The i386 version 'as' supports both the original Intel 386 architecture in both 16 and 32-bit mode as well as AMD x86-64 architecture extending the Intel architecture to 64-bits.

* Menu:

* i386-Options::	Options
* i386-Directives::	X86 specific directives
* i386-Syntax::	Syntactical considerations
* i386-Mnemonics::	Instruction Naming
* i386-Regs::	Register Naming
* i386-Prefixes::	Instruction Prefixes
* i386-Memory::	Memory References
* i386-Jumps::	Handling of Jump Instructions
* i386-Float::	Floating Point
* i386-SIMD::	Intel's MMX and AMD's 3DNow! SIMD Operations
* i386-LWP::	AMD's Lightweight Profiling Instructions
* i386-BMI::	Bit Manipulation Instruction
* i386-TBM::	AMD's Trailing Bit Manipulation Instructions
* i386-16bit::	Writing 16-bit Code
* i386-Arch::	Specifying an x86 CPU architecture
* i386-Bugs::	AT&T Syntax bugs
* i386-Notes::	Notes



File: as.info, Node: i386-Options, Next: i386-Directives, Up: i386-Dependent

9.15.1 Options

The i386 version of 'as' has a few machine dependent options:

'--32 | --x32 | --64'

Select the word size, either 32 bits or 64 bits. '--32' implies Intel i386 architecture, while '--x32' and '--64' imply AMD x86-64 architecture with 32-bit or 64-bit word-size respectively.

These options are only available with the ELF object file format, and require that the necessary BFD support has been included (on a 32-bit platform you have to add -enable-64-bit-bfd to configure enable 64-bit usage and use x86-64 as target platform).

'-n'

By default, x86 GAS replaces multiple nop instructions used for alignment within code sections with multi-byte nop instructions such as leal 0(%esi,1),%esi. This switch disables the optimization.

'--divide'

On SVR4-derived platforms, the character '/' is treated as a comment character, which means that it cannot be used in expressions. The '--divide' option turns '/' into a normal character. This does not disable '/' at the beginning of a line starting a comment, or affect using '#' for starting a comment.

'-march=CPU[+EXTENSION...]'

This option specifies the target processor. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target processor. The following processor names are recognized: 'i8086', 'i186', 'i286', 'i386', 'i486', 'i586', 'i686', 'pentium', 'pentiumpro', 'pentiumii', 'pentiumiii', 'pentium4', 'prescott', 'nocona', 'core', 'core2', 'corei7', 'l1om', 'k1om', 'iamcu', 'k6', 'k6_2', 'athlon', 'opteron', 'k8', 'amdfam10', 'bdver1', 'bdver2', 'bdver3', 'bdver4', 'znver1', 'btver1', 'btver2', 'generic32' and 'generic64'.

In addition to the basic instruction set, the assembler can be told to accept various extension mnemonics. For example,

'-march=i686+sse4+vmx' extends I686 with SSE4 and VMX. The following extensions are currently supported: '8087', '287', '387', '687', 'no87', 'no287', 'no387', 'no687', 'mmx', 'nommx', 'sse', 'sse2', 'sse3', 'ssse3', 'sse4.1', 'sse4.2', 'sse4', 'nosse', 'nosse2', 'nosse3', 'nossse3', 'nosse4.1', 'nosse4.2', 'nosse4', 'avx', 'avx2', 'noavx', 'noavx2', 'adx', 'rdseed', 'prfchw', 'smap', 'mpx', 'sha', 'rdpid', 'ptwrite', 'prefetchwt1', 'clflushopt', 'se1', 'clwb', 'avx512f', 'avx512cd', 'avx512er', 'avx512pf', 'avx512vl', 'avx512bw', 'avx512dq', 'avx512ifma', 'avx512vbmi', 'avx512_4fmaps', 'avx512_4vnniw', 'avx512_vpopcntdq', 'noavx512f', 'noavx512cd', 'noavx512er', 'noavx512pf', 'noavx512vl', 'noavx512bw', 'noavx512dq', 'noavx512ifma', 'noavx512vbmi', 'noavx512_4fmaps', 'noavx512_4vnniw', 'noavx512_vpopcntdq', 'vmx', 'vmfunc', 'smx', 'xsave', 'xsaveopt',

'xsavec', 'xsaves', 'aes', 'pclmul', 'fsgsbase', 'rdrnd', 'f16c', 'bmi2', 'fma', 'movbe', 'ept', 'lzcnt', 'hle', 'rtm', 'invpcid', 'clflush', 'mwaitx', 'clzero', 'lwp', 'fma4', 'xop', 'cx16', 'syscall', 'rdtscp', '3dnow', '3dnowa', 'sse4a', 'sse5', 'svme', 'abm' and 'padlock'. Note that rather than extending a basic instruction set, the extension mnemonics starting with 'no' revoke the respective functionality.

When the '.arch' directive is used with '-march', the '.arch' directive will take precedent.

'-mtune=CPU'

This option specifies a processor to optimize for. When used in conjunction with the '-march' option, only instructions of the processor specified by the '-march' option will be generated.

Valid CPU values are identical to the processor list of '-march=CPU'.

'-msse2avx'

This option specifies that the assembler should encode SSE instructions with VEX prefix.

'-msse-check=NONE'

'-msse-check=WARNING'

'-msse-check=ERROR'

These options control if the assembler should check SSE instructions. '-msse-check=NONE' will make the assembler not to check SSE instructions, which is the default.

'-msse-check=WARNING' will make the assembler issue a warning for any SSE instruction. '-msse-check=ERROR' will make the assembler issue an error for any SSE instruction.

'-mavxscalar=128'

'-mavxscalar=256'

These options control how the assembler should encode scalar AVX instructions. '-mavxscalar=128' will encode scalar AVX instructions with 128bit vector length, which is the default.

'-mavxscalar=256' will encode scalar AVX instructions with 256bit vector length.

'-mevexlig=128'

'-mevexlig=256'

'-mevexlig=512'

These options control how the assembler should encode length-ignored (LIG) EVEX instructions. '-mevexlig=128' will encode LIG EVEX instructions with 128bit vector length, which is the default. '-mevexlig=256' and '-mevexlig=512' will encode LIG EVEX instructions with 256bit and 512bit vector length, respectively.

'-mevexwig=0'

'-mevexwig=1'

These options control how the assembler should encode w-ignored (WIG) EVEX instructions. '-mevexwig=0' will encode WIG EVEX instructions with evex.w = 0, which is the default. '-mevexwig=1' will encode WIG EVEX instructions with evex.w = 1.

'-mmnemonic=ATT'

'-mmnemonic=INTEL'

This option specifies instruction mnemonic for matching instructions. The '.att_mnemonic' and '.intel_mnemonic' directives will take precedent.

'-msyntax=ATT'

'-msyntax=INTEL'

This option specifies instruction syntax when processing instructions. The '.att_syntax' and '.intel_syntax' directives will take precedent.

'-mnaked-reg'

This option specifies that registers don't require a '%' prefix. The '.att_syntax' and '.intel_syntax' directives will take precedent.

'-madd-bnd-prefix'

This option forces the assembler to add BND prefix to all branches, even if such prefix was not explicitly specified in the source code.

'-mno-shared'

On ELF target, the assembler normally optimizes out non-PLT relocations against defined non-weak global branch targets with default visibility. The '-mshared' option tells the assembler to generate code which may go into a shared library where all non-weak global branch targets with default visibility can be preempted. The resulting code is slightly bigger. This option only affects the handling of branch instructions.

'-mbig-obj'

On x86-64 PE/COFF target this option forces the use of big object file format, which allows more than 32768 sections.

'-momit-lock-prefix=NO'

'-momit-lock-prefix=YES'

These options control how the assembler should encode lock prefix. This option is intended as a workaround for processors, that fail on lock prefix. This option can only be safely used with single-core, single-thread computers '-momit-lock-prefix=YES' will omit all lock prefixes. '-momit-lock-prefix=NO' will encode lock prefix as usual, which is the default.

'-mfence-as-lock-add=NO'

'-mfence-as-lock-add=YES'

These options control how the assembler should encode lfence, mfence and sfence. '-mfence-as-lock-add=YES' will encode lfence, mfence and sfence as 'lock addl \$0x0, (%rsp)' in 64-bit mode and 'lock addl \$0x0, (%esp)' in 32-bit mode. '-mfence-as-lock-add=NO' will encode lfence, mfence and sfence as usual, which is the default.

'-mrelax-relocations=NO'

'-mrelax-relocations=YES'

These options control whether the assembler should generate relax relocations, R_386_GOT32X, in 32-bit mode, or R_X86_64_GOTPCRELX and R_X86_64_REX_GOTPCRELX, in 64-bit mode.

'-mrelax-relocations=YES' will generate relax relocations.

'-mrelax-relocations=NO' will not generate relax relocations. The

default can be controlled by a configure option
'--enable-x86-relax-relocations'.

'-mevexrcig=RNE'
'-mevexrcig=RD'
'-mevexrcig=RU'
'-mevexrcig=RZ'

These options control how the assembler should encode SAE-only EVEX instructions. '-mevexrcig=RNE' will encode RC bits of EVEX instruction with 00, which is the default. '-mevexrcig=RD', '-mevexrcig=RU' and '-mevexrcig=RZ' will encode SAE-only EVEX instructions with 01, 10 and 11 RC bits, respectively.

'-mamd64'
'-mintel64'

This option specifies that the assembler should accept only AMD64 or Intel64 ISA in 64-bit mode. The default is to accept both.



File: as.info, Node: i386-Directives, Next: i386-Syntax, Prev: i386-Options, Up: i386-Dependent

9.15.2 x86 specific Directives

'`.lcomm SYMBOL , LENGTH[, ALIGNMENT]`'

Reserve LENGTH (an absolute expression) bytes for a local common denoted by SYMBOL. The section and value of SYMBOL are those of the new local common. The addresses are allocated in the bss section, so that at run-time the bytes start off zeroed. Since SYMBOL is not declared global, it is normally not visible to 'ld'. The optional third parameter, ALIGNMENT, specifies the desired alignment of the symbol in the bss section.

This directive is only available for COFF based x86 targets.



File: as.info, Node: i386-Syntax, Next: i386-Mnemonics, Prev: i386-Directives, Up: i386-Dependent

9.15.3 i386 Syntactical Considerations

* Menu:

* i386-Variations:: AT&T Syntax versus Intel Syntax
* i386-Chars:: Special Characters



File: as.info, Node: i386-Variations, Next: i386-Chars, Up: i386-Syntax

9.15.3.1 AT&T Syntax versus Intel Syntax

.....

'as' now supports assembly using Intel assembler syntax. '`.intel_syntax`' selects Intel mode, and '`.att_syntax`' switches back to the usual AT&T mode for compatibility with the output of 'gcc'. Either of these directives may have an optional argument, 'prefix', or 'noprefix' specifying whether registers require a '%' prefix. AT&T

System V/386 assembler syntax is quite different from Intel syntax. We mention these differences because almost all 80386 documents use Intel syntax. Notable differences between the two syntaxes are:

- * AT&T immediate operands are preceded by '\$'; Intel immediate operands are undelimited (Intel 'push 4' is AT&T 'pushl \$4'). AT&T register operands are preceded by '%'; Intel register operands are undelimited. AT&T absolute (as opposed to PC relative) jump/call operands are prefixed by '*'; they are undelimited in Intel syntax.
- * AT&T and Intel syntax use the opposite order for source and destination operands. Intel 'add eax, 4' is 'addl \$4, %eax'. The 'source, dest' convention is maintained for compatibility with previous Unix assemblers. Note that 'bound', 'invlpga', and instructions with 2 immediate operands, such as the 'enter' instruction, do not have reversed order. *note i386-Bugs::.
- * In AT&T syntax the size of memory operands is determined from the last character of the instruction mnemonic. Mnemonic suffixes of 'b', 'w', 'l' and 'q' specify byte (8-bit), word (16-bit), long (32-bit) and quadruple word (64-bit) memory references. Intel syntax accomplishes this by prefixing memory operands (not the instruction mnemonics) with 'byte ptr', 'word ptr', 'dword ptr' and 'qword ptr'. Thus, Intel 'mov al, byte ptr F00' is 'movb F00, %al' in AT&T syntax.

In 64-bit code, 'movabs' can be used to encode the 'mov' instruction with the 64-bit displacement or immediate operand.

- * Immediate form long jumps and calls are 'lcall/ljmp \$SECTION, \$OFFSET' in AT&T syntax; the Intel syntax is 'call/jmp far SECTION:OFFSET'. Also, the far return instruction is 'lret \$STACK-ADJUST' in AT&T syntax; Intel syntax is 'ret far STACK-ADJUST'.
- * The AT&T assembler does not provide support for multiple section programs. Unix style systems expect all programs to be single sections.



File: as.info, Node: i386-Chars, Prev: i386-Variations, Up: i386-Syntax

9.15.3.2 Special Characters

.....

The presence of a '#' appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

If the '--divide' command line option has not been specified then the '/' character appearing anywhere on a line also introduces a line comment.

The ';' character can be used to separate statements on the same line.



File: as.info, Node: i386-Mnemonics, Next: i386-Regs, Prev: i386-Syntax, Up: i386-Dependent

9.15.4 i386-Mnemonics

9.15.4.1 Instruction Naming

.....

Instruction mnemonics are suffixed with one character modifiers which specify the size of operands. The letters 'b', 'w', 'l' and 'q' specify byte, word, long and quadruple word operands. If no suffix is specified by an instruction then 'as' tries to fill in the missing suffix based on the destination register operand (the last one by convention). Thus, 'mov %ax, %bx' is equivalent to 'movw %ax, %bx'; also, 'mov \$1, %bx' is equivalent to 'movw \$1, %bx'. Note that this is incompatible with the AT&T Unix assembler which assumes that a missing mnemonic suffix implies long operand size. (This incompatibility does not affect compiler output since compilers always explicitly specify the mnemonic suffix.)

Almost all instructions have the same names in AT&T and Intel format. There are a few exceptions. The sign extend and zero extend instructions need two sizes to specify them. They need a size to sign/zero extend from and a size to zero extend to. This is accomplished by using two instruction mnemonic suffixes in AT&T syntax. Base names for sign extend and zero extend are 'movs...' and 'movz...' in AT&T syntax ('movsx' and 'movzx' in Intel syntax). The instruction mnemonic suffixes are tacked on to this base name, the from suffix before the to suffix. Thus, 'movsbl %al, %edx' is AT&T syntax for "move sign extend from %al to %edx." Possible suffixes, thus, are 'bl' (from byte to long), 'bw' (from byte to word), 'wl' (from word to long), 'bq' (from byte to quadruple word), 'wq' (from word to quadruple word), and 'lq' (from long to quadruple word).

Different encoding options can be specified via optional mnemonic suffix. '.s' suffix swaps 2 register operands in encoding when moving from one register to another. '.d8' or '.d32' suffix prefers 8bit or 32bit displacement in encoding.

The Intel-syntax conversion instructions

- * 'cbw' -- sign-extend byte in '%al' to word in '%ax',
- * 'cwde' -- sign-extend word in '%ax' to long in '%eax',
- * 'cwd' -- sign-extend word in '%ax' to long in '%dx:%ax',
- * 'cdq' -- sign-extend dword in '%eax' to quad in '%edx:%eax',
- * 'cdqe' -- sign-extend dword in '%eax' to quad in '%rax' (x86-64 only),
- * 'cqo' -- sign-extend quad in '%rax' to octuple in '%rdx:%rax' (x86-64 only),

are called 'cbtw', 'cwtl', 'cwtl', 'cltd', 'cltq', and 'cqto' in AT&T naming. 'as' accepts either naming for these instructions.

Far call/jump instructions are 'lcall' and 'ljmp' in AT&T syntax, but are 'call far' and 'jump far' in Intel convention.

9.15.4.2 AT&T Mnemonic versus Intel Mnemonic

.....

'as' supports assembly using Intel mnemonic. '.intel_mnemonic' selects Intel mnemonic with Intel syntax, and '.att_mnemonic' switches back to the usual AT&T mnemonic with AT&T syntax for compatibility with the output of 'gcc'. Several x87 instructions, 'fadd', 'fdiv', 'fdivp', 'fdivr', 'fdivrp', 'fmul', 'fsub', 'fsubp', 'fsubr' and 'fsubrp', are implemented in AT&T System V/386 assembler with different mnemonics from those in Intel IA32 specification. 'gcc' generates those instructions with AT&T mnemonic.

US

File: as.info, Node: i386-Regs, Next: i386-Prefixes, Prev: i386-Mnemonics, Up: i386-Dependent

9.15.5 Register Naming

Register operands are always prefixed with '%'. The 80386 registers consist of

- * the 8 32-bit registers '%eax' (the accumulator), '%ebx', '%ecx', '%edx', '%edi', '%esi', '%ebp' (the frame pointer), and '%esp' (the stack pointer).
- * the 8 16-bit low-ends of these: '%ax', '%bx', '%cx', '%dx', '%di', '%si', '%bp', and '%sp'.
- * the 8 8-bit registers: '%ah', '%al', '%bh', '%bl', '%ch', '%cl', '%dh', and '%dl' (These are the high-bytes and low-bytes of '%ax', '%bx', '%cx', and '%dx')
- * the 6 section registers '%cs' (code section), '%ds' (data section), '%ss' (stack section), '%es', '%fs', and '%gs'.
- * the 5 processor control registers '%cr0', '%cr2', '%cr3', '%cr4', and '%cr8'.
- * the 6 debug registers '%db0', '%db1', '%db2', '%db3', '%db6', and '%db7'.
- * the 2 test registers '%tr6' and '%tr7'.
- * the 8 floating point register stack '%st' or equivalently '%st(0)', '%st(1)', '%st(2)', '%st(3)', '%st(4)', '%st(5)', '%st(6)', and '%st(7)'. These registers are overloaded by 8 MMX registers '%mm0', '%mm1', '%mm2', '%mm3', '%mm4', '%mm5', '%mm6' and '%mm7'.
- * the 8 128-bit SSE registers registers '%xmm0', '%xmm1', '%xmm2', '%xmm3', '%xmm4', '%xmm5', '%xmm6' and '%xmm7'.

The AMD x86-64 architecture extends the register set by:

- * enhancing the 8 32-bit registers to 64-bit: '%rax' (the

accumulator), '%rbx', '%rcx', '%rdx', '%rdi', '%rsi', '%rbp' (the frame pointer), '%rsp' (the stack pointer)

- * the 8 extended registers '%r8'-'%r15'.
- * the 8 32-bit low ends of the extended registers: '%r8d'-'%r15d'.
- * the 8 16-bit low ends of the extended registers: '%r8w'-'%r15w'.
- * the 8 8-bit low ends of the extended registers: '%r8b'-'%r15b'.
- * the 4 8-bit registers: '%sil', '%dil', '%bpl', '%spl'.
- * the 8 debug registers: '%db8'-'%db15'.
- * the 8 128-bit SSE registers: '%xmm8'-'%xmm15'.

With the AVX extensions more registers were made available:

- * the 16 256-bit SSE '%ymm0'-'%ymm15' (only the first 8 available in 32-bit mode). The bottom 128 bits are overlaid with the '%xmm0'-'%xmm15' registers.

The AVX2 extensions made in 64-bit mode more registers available:

- * the 16 128-bit registers '%xmm16'-'%xmm31' and the 16 256-bit registers '%ymm16'-'%ymm31'.

The AVX512 extensions added the following registers:

- * the 32 512-bit registers '%zmm0'-'%zmm31' (only the first 8 available in 32-bit mode). The bottom 128 bits are overlaid with the '%xmm0'-'%xmm31' registers and the first 256 bits are overlaid with the '%ymm0'-'%ymm31' registers.
- * the 8 mask registers '%k0'-'%k7'.



File: as.info, Node: i386-Prefixes, Next: i386-Memory, Prev: i386-Regs, Up: i386-Dependent

9.15.6 Instruction Prefixes

Instruction prefixes are used to modify the following instruction. They are used to repeat string instructions, to provide section overrides, to perform bus lock operations, and to change operand and address sizes. (Most instructions that normally operate on 32-bit operands will use 16-bit operands if the instruction has an "operand size" prefix.) Instruction prefixes are best written on the same line as the instruction they act upon. For example, the 'scas' (scan string) instruction is repeated with:

```
repne scas %es:(%edi),%al
```

You may also place prefixes on the lines immediately preceding the instruction, but this circumvents checks that 'as' does with prefixes, and will not work with all prefixes.

Here is a list of instruction prefixes:

- * Section override prefixes 'cs', 'ds', 'ss', 'es', 'fs', 'gs'. These are automatically added by specifying using the SECTION:MEMORY-OPERAND form for memory references.
- * Operand/Address size prefixes 'data16' and 'addr16' change 32-bit operands/addresses into 16-bit operands/addresses, while 'data32' and 'addr32' change 16-bit ones (in a '.code16' section) into 32-bit operands/addresses. These prefixes must appear on the same line of code as the instruction they modify. For example, in a 16-bit '.code16' section, you might write:


```
addr32 jmp1 *(%ebx)
```
- * The bus lock prefix 'lock' inhibits interrupts during execution of the instruction it precedes. (This is only valid with certain instructions; see a 80386 manual for details).
- * The wait for coprocessor prefix 'wait' waits for the coprocessor to complete the current instruction. This should never be needed for the 80386/80387 combination.
- * The 'rep', 'repe', and 'repne' prefixes are added to string instructions to make them repeat '%ecx' times ('%cx' times if the current address size is 16-bits).
- * The 'rex' family of prefixes is used by x86-64 to encode extensions to i386 instruction set. The 'rex' prefix has four bits -- an operand size overwrite ('64') used to change operand size from 32-bit to 64-bit and X, Y and Z extensions bits used to extend the register set.

You may write the 'rex' prefixes directly. The 'rex64xyz' instruction emits 'rex' prefix with all the bits set. By omitting the '64', 'x', 'y' or 'z' you may write other prefixes as well. Normally, there is no need to write the prefixes explicitly, since gas will automatically generate them based on the instruction operands.



File: as.info, Node: i386-Memory, Next: i386-Jumps, Prev: i386-Prefixes, Up: i386-Dependent

9.15.7 Memory References

An Intel syntax indirect memory reference of the form

```
SECTION:[BASE + INDEX*SCALE + DISP]
```

is translated into the AT&T syntax

```
SECTION:DISP(BASE, INDEX, SCALE)
```

where BASE and INDEX are the optional 32-bit base and index registers, DISP is the optional displacement, and SCALE, taking the values 1, 2, 4, and 8, multiplies INDEX to calculate the address of the operand. If no SCALE is specified, SCALE is taken to be 1. SECTION specifies the optional section register for the memory operand, and may override the

default section register (see a 80386 manual for section register defaults). Note that section overrides in AT&T syntax `_must_` be preceded by a '%'. If you specify a section override which coincides with the default section register, 'as' does `_not_` output any section register override prefixes to assemble the given instruction. Thus, section overrides can be specified to emphasize which section register is used for a given memory operand.

Here are some examples of Intel and AT&T style memory references:

AT&T: '-4(%ebp)', Intel: '[ebp - 4]'
 BASE is '%ebp'; DISP is '-4'. SECTION is missing, and the default section is used ('%ss' for addressing with '%ebp' as the base register). INDEX, SCALE are both missing.

AT&T: 'foo(,%eax,4)', Intel: '[foo + eax*4]'
 INDEX is '%eax' (scaled by a SCALE 4); DISP is 'foo'. All other fields are missing. The section register here defaults to '%ds'.

AT&T: 'foo(,1)'; Intel '[foo]'
 This uses the value pointed to by 'foo' as a memory operand. Note that BASE and INDEX are both missing, but there is only `_one_` ', '. This is a syntactic exception.

AT&T: '%gs:foo'; Intel 'gs:foo'
 This selects the contents of the variable 'foo' with section register SECTION being '%gs'.

Absolute (as opposed to PC relative) call and jump operands must be prefixed with '*'. If no '*' is specified, 'as' always chooses PC relative addressing for jump/call labels.

Any instruction that has a memory operand, but no register operand, `_must_` specify its size (byte, word, long, or quadruple) with an instruction mnemonic suffix ('b', 'w', 'l' or 'q', respectively).

The x86-64 architecture adds an RIP (instruction pointer relative) addressing. This addressing mode is specified by using 'rip' as a base register. Only constant offsets are valid. For example:

AT&T: '1234(%rip)', Intel: '[rip + 1234]'
 Points to the address 1234 bytes past the end of the current instruction.

AT&T: 'symbol(%rip)', Intel: '[rip + symbol]'
 Points to the 'symbol' in RIP relative way, this is shorter than the default absolute addressing.

Other addressing modes remain unchanged in x86-64 architecture, except registers used are 64-bit instead of 32-bit.



File: as.info, Node: i386-Jumps, Next: i386-Float, Prev: i386-Memory, Up: i386-Dependent

9.15.8 Handling of Jump Instructions

Jump instructions are always optimized to use the smallest possible

displacements. This is accomplished by using byte (8-bit) displacement jumps whenever the target is sufficiently close. If a byte displacement is insufficient a long displacement is used. We do not support word (16-bit) displacement jumps in 32-bit mode (i.e. prefixing the jump instruction with the 'data16' instruction prefix), since the 80386 insists upon masking '%eip' to 16 bits after the word displacement is added. (See also *note i386-Arch:.)

Note that the 'jcxz', 'jecxz', 'loop', 'loopz', 'loope', 'loopnz' and 'loopne' instructions only come in byte displacements, so that if you use these instructions ('gcc' does not use them) you may get an error message (and incorrect code). The AT&T 80386 assembler tries to get around this problem by expanding 'jcxz foo' to

```

                jcxz cx_zero
                jmp  cx_nonzero
cx_zero:  jmp  foo
cx_nonzero:

```



File: as.info, Node: i386-Float, Next: i386-SIMD, Prev: i386-Jumps, Up: i386-Dependent

9.15.9 Floating Point

All 80387 floating point types except packed BCD are supported. (BCD support may be added without much difficulty). These data types are 16-, 32-, and 64- bit integers, and single (32-bit), double (64-bit), and extended (80-bit) precision floating point. Each supported type has an instruction mnemonic suffix and a constructor associated with it. Instruction mnemonic suffixes specify the operand's data type. Constructors build these data types into memory.

- * Floating point constructors are '.float' or '.single', '.double', and '.tfloat' for 32-, 64-, and 80-bit formats. These correspond to instruction mnemonic suffixes 's', 'l', and 't'. 't' stands for 80-bit (ten byte) real. The 80387 only supports this format via the 'fldt' (load 80-bit real to stack top) and 'fstpt' (store 80-bit real and pop stack) instructions.
- * Integer constructors are '.word', '.long' or '.int', and '.quad' for the 16-, 32-, and 64-bit integer formats. The corresponding instruction mnemonic suffixes are 's' (single), 'l' (long), and 'q' (quad). As with the 80-bit real format, the 64-bit 'q' format is only present in the 'fildq' (load quad integer to stack top) and 'fistpq' (store quad integer and pop stack) instructions.

Register to register operations should not use instruction mnemonic suffixes. 'fstl %st, %st(1)' will give a warning, and be assembled as if you wrote 'fst %st, %st(1)', since all register to register operations use 80-bit floating point operands. (Contrast this with 'fstl %st, mem', which converts '%st' from 80-bit to 64-bit floating point format, then stores the result in the 4 byte location 'mem')



File: as.info, Node: i386-SIMD, Next: i386-LWP, Prev: i386-Float, Up: i386-Dependent

9.15.10 Intel's MMX and AMD's 3DNow! SIMD Operations

'as' supports Intel's MMX instruction set (SIMD instructions for integer data), available on Intel's Pentium MMX processors and Pentium II processors, AMD's K6 and K6-2 processors, Cyrix' M2 processor, and probably others. It also supports AMD's 3DNow! instruction set (SIMD instructions for 32-bit floating point data) available on AMD's K6-2 processor and possibly others in the future.

Currently, 'as' does not support Intel's floating point SIMD, Katmai (KNI).

The eight 64-bit MMX operands, also used by 3DNow!, are called '%mm0', '%mm1', ... '%mm7'. They contain eight 8-bit integers, four 16-bit integers, two 32-bit integers, one 64-bit integer, or two 32-bit floating point values. The MMX registers cannot be used at the same time as the floating point stack.

See Intel and AMD documentation, keeping in mind that the operand order in instructions is reversed from the Intel syntax.



File: as.info, Node: i386-LWP, Next: i386-BMI, Prev: i386-SIMD, Up: i386-Dependent

9.15.11 AMD's Lightweight Profiling Instructions

'as' supports AMD's Lightweight Profiling (LWP) instruction set, available on AMD's Family 15h (Orochi) processors.

LWP enables applications to collect and manage performance data, and react to performance events. The collection of performance data requires no context switches. LWP runs in the context of a thread and so several counters can be used independently across multiple threads. LWP can be used in both 64-bit and legacy 32-bit modes.

For detailed information on the LWP instruction set, see the 'AMD Lightweight Profiling Specification' available at Lightweight Profiling Specification (<http://developer.amd.com/cpu/LWP>).



File: as.info, Node: i386-BMI, Next: i386-TBM, Prev: i386-LWP, Up: i386-Dependent

9.15.12 Bit Manipulation Instructions

'as' supports the Bit Manipulation (BMI) instruction set.

BMI instructions provide several instructions implementing individual bit manipulation operations such as isolation, masking, setting, or resetting.



File: as.info, Node: i386-TBM, Next: i386-16bit, Prev: i386-BMI, Up: i386-Dependent

9.15.13 AMD's Trailing Bit Manipulation Instructions

'as' supports AMD's Trailing Bit Manipulation (TBM) instruction set, available on AMD's BDVER2 processors (Trinity and Viperfish).

TBM instructions provide instructions implementing individual bit manipulation operations such as isolating, masking, setting, resetting, complementing, and operations on trailing zeros and ones.

US

File: as.info, Node: i386-16bit, Next: i386-Arch, Prev: i386-TBM, Up: i386-Dependent

9.15.14 Writing 16-bit Code

While 'as' normally writes only "pure" 32-bit i386 code or 64-bit x86-64 code depending on the default configuration, it also supports writing code to run in real mode or in 16-bit protected mode code segments. To do this, put a `.code16` or `.code16gcc` directive before the assembly language instructions to be run in 16-bit mode. You can switch 'as' to writing 32-bit code with the `.code32` directive or 64-bit code with the `.code64` directive.

`.code16gcc` provides experimental support for generating 16-bit code from gcc, and differs from `.code16` in that `call`, `ret`, `enter`, `leave`, `push`, `pop`, `pusha`, `popa`, `pushf`, and `popf` instructions default to 32-bit size. This is so that the stack pointer is manipulated in the same way over function calls, allowing access to function parameters at the same stack offsets as in 32-bit mode. `.code16gcc` also automatically adds address size prefixes where necessary to use the 32-bit addressing modes that gcc generates.

The code which 'as' generates in 16-bit mode will not necessarily run on a 16-bit pre-80386 processor. To write code that runs on such a processor, you must refrain from using `_any_` 32-bit constructs which require 'as' to output address or operand size prefixes.

Note that writing 16-bit code instructions by explicitly specifying a prefix or an instruction mnemonic suffix within a 32-bit code section generates different machine instructions than those generated for a 16-bit code segment. In a 32-bit code section, the following code generates the machine opcode bytes `'66 6a 04'`, which pushes the value `'4'` onto the stack, decrementing `'%esp'` by 2.

```
pushw $4
```

The same code in a 16-bit code section would generate the machine opcode bytes `'6a 04'` (i.e., without the operand size prefix), which is correct since the processor default operand size is assumed to be 16 bits in a 16-bit code section.

US

File: as.info, Node: i386-Arch, Next: i386-Bugs, Prev: i386-16bit, Up: i386-Dependent

9.15.15 Specifying CPU Architecture

'as' may be told to assemble for a particular CPU (sub-)architecture with the `.arch CPU_TYPE` directive. This directive enables a warning when gas detects an instruction that is not supported on the CPU specified. The choices for `CPU_TYPE` are:

'i8086'	'i186'	'i286'	'i386'
'i486'	'i586'	'i686'	'pentium'
'pentiumpro'	'pentiumii'	'pentiumiii'	'pentium4'
'prescott'	'nocona'	'core'	'core2'
'corei7'	'l1om'	'k1om'	
		'iamcu'	
'k6'	'k6_2'	'athlon'	'k8'
'amdfam10'	'bdver1'	'bdver2'	'bdver3'
'bdver4'	'znver1'	'btver1'	'btver2'
'generic32'	'generic64'		
'mmx'	'sse'	'sse2'	'sse3'
'ssse3'	'sse4.1'	'sse4.2'	'sse4'
'avx'	'vmx'	'smx'	'ept'
'clflush'	'movbe'	'xsave'	'xsaveopt'
'aes'	'pclmul'	'fma'	'fsgsbase'
'rdrnd'	'f16c'	'avx2'	'bmi2'
'lzcnt'	'invpcid'	'vmfunc'	'hle'
'rtm'	'adx'	'rdseed'	'prfchw'
'smap'	'mpx'	'sha'	'prefetchwt1'
'clflushopt'	'xsavc'	'xsaves'	'se1'
'avx512f'	'avx512cd'	'avx512er'	'avx512pf'
'avx512vl'	'avx512bw'	'avx512dq'	'avx512ifma'
'avx512vbmi'	'avx512_4fmaps'	'avx512_4vnniw'	
'avx512_vpopcntdq'	'clwb'	'rdpid'	'ptwrite'
'3dnow'	'3dnowa'	'sse4a'	'sse5'
'syscall'	'rdtscp'	'svme'	'abm'
'lwp'	'fma4'	'xop'	'cx16'
'padlock'	'clzero'	'mwaitx'	

Apart from the warning, there are only two other effects on 'as' operation; Firstly, if you specify a CPU other than 'i486', then shift by one instructions such as `sarl $1, %eax` will automatically use a two byte opcode sequence. The larger three byte opcode sequence is used on the 486 (and when no architecture is specified) because it executes faster on the 486. Note that you can explicitly request the two byte opcode by writing `sarl %eax`. Secondly, if you specify 'i8086', 'i186', or 'i286', `_and_ .code16` or `.code16gcc` then byte offset conditional jumps will be promoted when necessary to a two instruction sequence consisting of a conditional jump of the opposite sense around an unconditional jump to the target.

Following the CPU architecture (but not a sub-architecture, which are those starting with a dot), you may specify 'jumps' or 'nojumps' to control automatic promotion of conditional jumps. 'jumps' is the default, and enables jump promotion; All external jumps will be of the long variety, and file-local jumps will be promoted as necessary. (*note i386-Jumps::) 'nojumps' leaves external conditional jumps as byte offset jumps, and warns about file-local conditional jumps that 'as' promotes. Unconditional jumps are treated as for 'jumps'.

For example

```
.arch i8086,nojumps
```



File: as.info, Node: i386-Bugs, Next: i386-Notes, Prev: i386-Arch, Up: i386-Dependent

9.15.16 AT&T Syntax bugs

The UnixWare assembler, and probably other AT&T derived ix86 Unix assemblers, generate floating point instructions with reversed source and destination registers in certain cases. Unfortunately, gcc and possibly many other programs use this reversed syntax, so we're stuck with it.

For example

```
fsub %st,%st(3)
```

results in '%st(3)' being updated to '%st - %st(3)' rather than the expected '%st(3) - %st'. This happens with all the non-commutative arithmetic floating point operations with two register operands where the source register is '%st' and the destination register is '%st(i)'.



File: as.info, Node: i386-Notes, Prev: i386-Bugs, Up: i386-Dependent

9.15.17 Notes

There is some trickery concerning the 'mul' and 'imul' instructions that deserves mention. The 16-, 32-, 64- and 128-bit expanding multiplies (base opcode '0xf6'; extension 4 for 'mul' and 5 for 'imul') can be output only in the one operand form. Thus, 'imul %ebx, %eax' does not select the expanding multiply; the expanding multiply would clobber the '%edx' register, and this would confuse 'gcc' output. Use 'imul %ebx' to get the 64-bit product in '%edx:%eax'.

We have added a two operand form of 'imul' when the first operand is an immediate mode expression and the second operand is a register. This is just a shorthand, so that, multiplying '%eax' by 69, for example, can be done with 'imul \$69, %eax' rather than 'imul \$69, %eax, %eax'.



File: as.info, Node: i860-Dependent, Next: i960-Dependent, Prev: i386-Dependent, Up: Machine Dependencies

9.16 Intel i860 Dependent Features

=====

* Menu:

```
* Notes-i860::                i860 Notes
* Options-i860::              i860 Command-line Options
* Directives-i860::          i860 Machine Directives
* Opcodes for i860::         i860 Opcodes
* Syntax of i860::           i860 Syntax
```



File: as.info, Node: Notes-i860, Next: Options-i860, Up: i860-Dependent

9.16.1 i860 Notes

 This is a fairly complete i860 assembler which is compatible with the UNIX System V/860 Release 4 assembler. However, it does not currently support SVR4 PIC (i.e., '@GOT, @GOTOFF, @PLT').

Like the SVR4/860 assembler, the output object format is ELF32. Currently, this is the only supported object format. If there is sufficient interest, other formats such as COFF may be implemented.

Both the Intel and AT&T/SVR4 syntaxes are supported, with the latter being the default. One difference is that AT&T syntax requires the '%' prefix on register names while Intel syntax does not. Another difference is in the specification of relocatable expressions. The Intel syntax is 'ha%expression' whereas the SVR4 syntax is '[expression]@ha' (and similarly for the "l" and "h" selectors).



File: as.info, Node: Options-i860, Next: Directives-i860, Prev: Notes-i860, Up: i860-Dependent

9.16.2 i860 Command-line Options

9.16.2.1 SVR4 compatibility options

.....

'-V'
 Print assembler version.
 '-Qy'
 Ignored.
 '-Qn'
 Ignored.

9.16.2.2 Other options

.....

'-EL'
 Select little endian output (this is the default).
 '-EB'
 Select big endian output. Note that the i860 always reads instructions as little endian data, so this option only effects data and not instructions.
 '-mwarn-expand'
 Emit a warning message if any pseudo-instruction expansions occurred. For example, a 'or' instruction with an immediate larger than 16-bits will be expanded into two instructions. This is a very undesirable feature to rely on, so this flag can help detect any code where it happens. One use of it, for instance, has been to find and eliminate any place where 'gcc' may emit these pseudo-instructions.
 '-mXP'
 Enable support for the i860XP instructions and control registers. By default, this option is disabled so that only the base instruction set (i.e., i860XR) is supported.
 '-mintel-syntax'
 The i860 assembler defaults to AT&T/SVR4 syntax. This option enables the Intel syntax.



File: as.info, Node: Directives-i860, Next: Opcodes for i860, Prev: Options-i860, Up: i860-Dependent

9.16.3 i860 Machine Directives

'`.dual`'

Enter dual instruction mode. While this directive is supported, the preferred way to use dual instruction mode is to explicitly code the dual bit with the '`d.`' prefix.

'`.enddual`'

Exit dual instruction mode. While this directive is supported, the preferred way to use dual instruction mode is to explicitly code the dual bit with the '`d.`' prefix.

'`.atmp`'

Change the temporary register used when expanding pseudo operations. The default register is '`r31`'.

The '`.dual`', '`.enddual`', and '`.atmp`' directives are available only in the Intel syntax mode.

Both syntaxes allow for the standard '`.align`' directive. However, the Intel syntax additionally allows keywords for the alignment parameter: "`.align type`", where '`type`' is one of '`.short`', '`.long`', '`.quad`', '`.single`', '`.double`' representing alignments of 2, 4, 16, 4, and 8, respectively.



File: as.info, Node: Opcodes for i860, Next: Syntax of i860, Prev: Directives-i860, Up: i860-Dependent

9.16.4 i860 Opcodes

All of the Intel i860XR and i860XP machine instructions are supported. Please see either `_i860 Microprocessor Programmer's Reference Manual_` or `_i860 Microprocessor Architecture_` for more information.

9.16.4.1 Other instruction support (pseudo-instructions)

.....

For compatibility with some other i860 assemblers, a number of pseudo-instructions are supported. While these are supported, they are a very undesirable feature that should be avoided - in particular, when they result in an expansion to multiple actual i860 instructions. Below are the pseudo-instructions that result in expansions.

* Load large immediate into general register:

The pseudo-instruction '`mov imm,%rn`' (where the immediate does not fit within a signed 16-bit field) will be expanded into:

```
orh large_imm@h,%r0,%rn
or large_imm@l,%rn,%rn
```

* Load/store with relocatable address expression:

For example, the pseudo-instruction '`ld.b addr_exp(%rx),%rn`' will be expanded into:

* Menu:

```
* Options-i960::          i960 Command-line Options
* Floating Point-i960::   Floating Point
* Directives-i960::      i960 Machine Directives
* Opcodes for i960::     i960 Opcodes
* Syntax of i960::       i960 Syntax
```

US

File: as.info, Node: Options-i960, Next: Floating Point-i960, Up: i960-Dependent

9.17.1 i960 Command-line Options

'-ACA | -ACA_A | -ACB | -ACC | -AKA | -AKB | -AKC | -AMC'

Select the 80960 architecture. Instructions or features not supported by the selected architecture cause fatal errors.

'-ACA' is equivalent to '-ACA_A'; '-AKC' is equivalent to '-AMC'. Synonyms are provided for compatibility with other tools.

If you do not specify any of these options, 'as' generates code for any instruction or feature that is supported by some version of the 960 (even if this means mixing architectures!). In principle, 'as' attempts to deduce the minimal sufficient processor type if none is specified; depending on the object code format, the processor type may be recorded in the object file. If it is critical that the 'as' output match a specific architecture, specify that architecture explicitly.

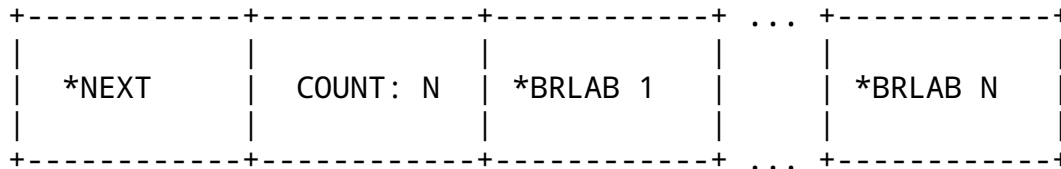
'-b'

Add code to collect information about conditional branches taken, for later optimization using branch prediction bits. (The conditional branch instructions have branch prediction bits in the CA, CB, and CC architectures.) If BR represents a conditional branch instruction, the following represents the code generated by the assembler when '-b' is specified:

```
          call    INCREMENT ROUTINE
          .word   0          # pre-counter
Label:   BR
          call    INCREMENT ROUTINE
          .word   0          # post-counter
```

The counter following a branch records the number of times that branch was not taken; the difference between the two counters is the number of times the branch was taken.

A table of every such 'Label' is also generated, so that the external postprocessor 'gbr960' (supplied by Intel) can locate all the counters. This table is always labeled '__BRANCH_TABLE__'; this is a local symbol to permit collecting statistics for many separate object files. The table is word aligned, and begins with a two-word header. The first word, initialized to 0, is used in maintaining linked lists of branch tables. The second word is a count of the number of entries in the table, which follow immediately: each is a word, pointing to one of the labels illustrated above.



__BRANCH_TABLE__ layout

The first word of the header is used to locate multiple branch tables, since each object file may contain one. Normally the links are maintained with a call to an initialization routine, placed at the beginning of each function in the file. The GNU C compiler generates these calls automatically when you give it a '-b' option. For further details, see the documentation of 'gbr960'.

'-no-relax'

Normally, Compare-and-Branch instructions with targets that require displacements greater than 13 bits (or that have external targets) are replaced with the corresponding compare (or 'chkbit') and branch instructions. You can use the '-no-relax' option to specify that 'as' should generate errors instead, if the target displacement is larger than 13 bits.

This option does not affect the Compare-and-Jump instructions; the code emitted for them is always adjusted when necessary (depending on displacement size), regardless of whether you use '-no-relax'.



File: as.info, Node: Floating Point-i960, Next: Directives-i960, Prev: Options-i960, Up: i960-Dependent

9.17.2 Floating Point

'as' generates IEEE floating-point numbers for the directives '.float', '.double', '.extended', and '.single'.



File: as.info, Node: Directives-i960, Next: Opcodes for i960, Prev: Floating Point-i960, Up: i960-Dependent

9.17.3 i960 Machine Directives

' .bss SYMBOL, LENGTH, ALIGN'

Reserve LENGTH bytes in the bss section for a local SYMBOL, aligned to the power of two specified by ALIGN. LENGTH and ALIGN must be positive absolute expressions. This directive differs from '.lcomm' only in that it permits you to specify an alignment.
*Note '.lcomm': Lcomm.

' .extended FLONUMS'

' .extended' expects zero or more flonums, separated by commas; for each flonum, '.extended' emits an IEEE extended-format (80-bit) floating-point number.

' .leafproc CALL-LAB, BAL-LAB'

You can use the `.leafproc` directive in conjunction with the optimized `callj` instruction to enable faster calls of leaf procedures. If a procedure is known to call no other procedures, you may define an entry point that skips procedure prolog code (and that does not depend on system-supplied saved context), and declare it as the BAL-LAB using `.leafproc`. If the procedure also has an entry point that goes through the normal prolog, you can specify that entry point as CALL-LAB.

A `.leafproc` declaration is meant for use in conjunction with the optimized call instruction `callj`; the directive records the data needed later to choose between converting the `callj` into a `bal` or a `call`.

CALL-LAB is optional; if only one argument is present, or if the two arguments are identical, the single argument is assumed to be the `bal` entry point.

`.sysproc NAME, INDEX`

The `.sysproc` directive defines a name for a system procedure. After you define it using `.sysproc`, you can use NAME to refer to the system procedure identified by INDEX when calling procedures with the optimized call instruction `callj`.

Both arguments are required; INDEX must be between 0 and 31 (inclusive).

US

File: as.info, Node: Opcodes for i960, Next: Syntax of i960, Prev: Directives-i960, Up: i960-Dependent

9.17.4 i960 Opcodes

All Intel 960 machine instructions are supported; *note i960
 Command-line Options: Options-i960. for a discussion of selecting the instruction subset for a particular 960 architecture.

Some opcodes are processed beyond simply emitting a single corresponding instruction: `callj`, and Compare-and-Branch or Compare-and-Jump instructions with target displacements larger than 13 bits.

* Menu:

* `callj-i960::` `callj`
 * `Compare-and-branch-i960::` `Compare-and-Branch`

US

File: as.info, Node: `callj-i960`, Next: `Compare-and-branch-i960`, Up: `Opcodes for i960`

9.17.4.1 `callj`

.....

You can write `callj` to have the assembler or the linker determine the most appropriate form of subroutine call: `call`, `bal`, or `calls`. If the assembly source contains enough information--a `.leafproc` or `.sysproc` directive defining the operand--then `as` translates the

'callj'; if not, it simply emits the 'callj', leaving it for the linker to resolve.



File: as.info, Node: Compare-and-branch-i960, Prev: callj-i960, Up: Opcodes for i960

9.17.4.2 Compare-and-Branch

.....

The 960 architectures provide combined Compare-and-Branch instructions that permit you to store the branch target in the lower 13 bits of the instruction word itself. However, if you specify a branch target far enough away that its address won't fit in 13 bits, the assembler can either issue an error, or convert your Compare-and-Branch instruction into separate instructions to do the compare and the branch.

Whether 'as' gives an error or expands the instruction depends on two choices you can make: whether you use the '-no-relax' option, and whether you use a "Compare and Branch" instruction or a "Compare and Jump" instruction. The "Jump" instructions are always expanded if necessary; the "Branch" instructions are expanded when necessary unless you specify '-no-relax'--in which case 'as' gives an error instead.

These are the Compare-and-Branch instructions, their "Jump" variants, and the instruction pairs they may expand into:

Compare and Branch	Jump	Expanded to
-----	-----	-----
bbc		chkbit; bno
bbs		chkbit; bo
cmpibe	cmpije	cmpi; be
cmpibg	cmpijg	cmpi; bg
cmpibge	cmpijge	cmpi; bge
cmpibl	cmpijl	cmpi; bl
cmpible	cmpijle	cmpi; ble
cmpibno	cmpijno	cmpi; bno
cmpibne	cmpijne	cmpi; bne
cmpibo	cmpijo	cmpi; bo
cmpobe	cmpoje	cmpo; be
cmpobg	cmpojg	cmpo; bg
cmpobge	cmpojge	cmpo; bge
cmpobl	cmpojl	cmpo; bl
cmpoble	cmpojle	cmpo; ble
cmpobne	cmpojne	cmpo; bne



File: as.info, Node: Syntax of i960, Prev: Opcodes for i960, Up: i960-Dependent

9.17.5 Syntax for the i960

* Menu:

* i960-Chars:: Special Characters



File: as.info, Node: i960-Chars, Up: Syntax of i960

9.17.5.1 Special Characters

.....

The presence of a '#' on a line indicates the start of a comment that extends to the end of the current line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The ';' character can be used to separate statements on the same line.

US

File: as.info, Node: IA-64-Dependent, Next: IP2K-Dependent, Prev: i960-Dependent, Up: Machine Dependencies

9.18 IA-64 Dependent Features

=====

* Menu:

* IA-64 Options::	Options
* IA-64 Syntax::	Syntax
* IA-64 Opcodes::	Opcodes

US

File: as.info, Node: IA-64 Options, Next: IA-64 Syntax, Up: IA-64-Dependent

9.18.1 Options

'-mconstant-gp'

This option instructs the assembler to mark the resulting object file as using the "constant GP" model. With this model, it is assumed that the entire program uses a single global pointer (GP) value. Note that this option does not in any fashion affect the machine code emitted by the assembler. All it does is turn on the EF_IA_64_CONS_GP flag in the ELF file header.

'-mauto-pic'

This option instructs the assembler to mark the resulting object file as using the "constant GP without function descriptor" data model. This model is like the "constant GP" model, except that it additionally does away with function descriptors. What this means is that the address of a function refers directly to the function's code entry-point. Normally, such an address would refer to a function descriptor, which contains both the code entry-point and the GP-value needed by the function. Note that this option does not in any fashion affect the machine code emitted by the assembler. All it does is turn on the EF_IA_64_NOFUNCDESC_CONS_GP flag in the ELF file header.

'-milp32'

'-milp64'

'-mlp64'

'-mp64'

These options select the data model. The assembler defaults to '-mlp64' (LP64 data model).

'-mle'

'-mbe'

These options select the byte order. The '-mle' option selects little-endian byte order (default) and '-mbe' selects big-endian byte order. Note that IA-64 machine code always uses little-endian byte order.

'-mtune=itanium1'

'-mtune=itanium2'

Tune for a particular IA-64 CPU, ITANIUM1 or ITANIUM2. The default is ITANIUM2.

'-munwind-check=warning'

'-munwind-check=error'

These options control what the assembler will do when performing consistency checks on unwind directives. '-munwind-check=warning' will make the assembler issue a warning when an unwind directive check fails. This is the default. '-munwind-check=error' will make the assembler issue an error when an unwind directive check fails.

'-mhint.b=ok'

'-mhint.b=warning'

'-mhint.b=error'

These options control what the assembler will do when the 'hint.b' instruction is used. '-mhint.b=ok' will make the assembler accept 'hint.b'. '-mhint.b=warning' will make the assembler issue a warning when 'hint.b' is used. '-mhint.b=error' will make the assembler treat 'hint.b' as an error, which is the default.

'-x'

'-xexplicit'

These options turn on dependency violation checking.

'-xauto'

This option instructs the assembler to automatically insert stop bits where necessary to remove dependency violations. This is the default mode.

'-xnone'

This option turns off dependency violation checking.

'-xdebug'

This turns on debug output intended to help tracking down bugs in the dependency violation checker.

'-xdebugn'

This is a shortcut for -xnone -xdebug.

'-xdebugx'

This is a shortcut for -xexplicit -xdebug.



9.18.2 Syntax

The assembler syntax closely follows the IA-64 Assembly Language Reference Guide.

* Menu:

```
* IA-64-Chars::          Special Characters
* IA-64-Regs::          Register Names
* IA-64-Bits::          Bit Names
* IA-64-Relocs::        Relocations
```



File: as.info, Node: IA-64-Chars, Next: IA-64-Regs, Up: IA-64 Syntax

9.18.2.1 Special Characters

.....

'//' is the line comment token.

';' can be used instead of a newline to separate statements.



File: as.info, Node: IA-64-Regs, Next: IA-64-Bits, Prev: IA-64-Chars, Up: IA-64 Syntax

9.18.2.2 Register Names

.....

The 128 integer registers are referred to as 'rN'. The 128 floating-point registers are referred to as 'fN'. The 128 application registers are referred to as 'arN'. The 128 control registers are referred to as 'crN'. The 64 one-bit predicate registers are referred to as 'pN'. The 8 branch registers are referred to as 'bN'. In addition, the assembler defines a number of aliases: 'gp' ('r1'), 'sp' ('r12'), 'rp' ('b0'), 'ret0' ('r8'), 'ret1' ('r9'), 'ret2' ('r10'), 'ret3' ('r9'), 'fargN' ('f8+N'), and 'fretN' ('f8+N').

For convenience, the assembler also defines aliases for all named application and control registers. For example, 'ar.bsp' refers to the register backing store pointer ('ar17'). Similarly, 'cr.eoi' refers to the end-of-interrupt register ('cr67').



File: as.info, Node: IA-64-Bits, Next: IA-64-Relocs, Prev: IA-64-Regs, Up: IA-64 Syntax

9.18.2.3 IA-64 Processor-Status-Register (PSR) Bit Names

.....

The assembler defines bit masks for each of the bits in the IA-64 processor status register. For example, 'psr.ic' corresponds to a value of 0x2000. These masks are primarily intended for use with the 'ssm'/'sum' and 'rsm'/'rum' instructions, but they can be used anywhere else where an integer constant is expected.



File: as.info, Node: IA-64-Relocs, Prev: IA-64-Bits, Up: IA-64 Syntax

9.18.2.4 Relocations

.....

In addition to the standard IA-64 relocations, the following relocations are implemented by 'as':

'@slotcount(V)'

Convert the address offset V into a slot count. This pseudo function is available only on VMS. The expression V must be known at assembly time: it can't reference undefined symbols or symbols in different sections.



File: as.info, Node: IA-64 Opcodes, Prev: IA-64 Syntax, Up: IA-64-Dependent

9.18.3 Opcodes

For detailed information on the IA-64 machine instruction set, see the IA-64 Architecture Handbook (http://developer.intel.com/design/itanium/arch_spec.htm).



File: as.info, Node: IP2K-Dependent, Next: LM32-Dependent, Prev: IA-64-Dependent, Up: Machine Dependencies

9.19 IP2K Dependent Features

=====

* Menu:

* IP2K-Opts:: IP2K Options
* IP2K-Syntax:: IP2K Syntax



File: as.info, Node: IP2K-Opts, Next: IP2K-Syntax, Up: IP2K-Dependent

9.19.1 IP2K Options

The Uvicom IP2K version of 'as' has a few machine dependent options:

'-mip2022ext'

'as' can assemble the extended IP2022 instructions, but it will only do so if this is specifically allowed via this command line option.

'-mip2022'

This option restores the assembler's default behaviour of not permitting the extended IP2022 instructions to be assembled.



File: as.info, Node: IP2K-Syntax, Prev: IP2K-Opts, Up: IP2K-Dependent

9.19.2 IP2K Syntax

Enable data cache related CSRs.

'-mbreak-enabled'

Enable break instructions.

'-mall-enabled'

Enable all instructions and CSRs.



File: as.info, Node: LM32 Syntax, Next: LM32 Opcodes, Prev: LM32 Options, Up: LM32-Dependent

9.20.2 Syntax

* Menu:

* LM32-Regs:: Register Names
 * LM32-Modifiers:: Relocatable Expression Modifiers
 * LM32-Chars:: Special Characters



File: as.info, Node: LM32-Regs, Next: LM32-Modifiers, Up: LM32 Syntax

9.20.2.1 Register Names

.....

LM32 has 32 x 32-bit general purpose registers 'r0', 'r1', ... 'r31'.

The following aliases are defined: 'gp' - 'r26', 'fp' - 'r27', 'sp' - 'r28', 'ra' - 'r29', 'ea' - 'r30', 'ba' - 'r31'.

LM32 has the following Control and Status Registers (CSRs).

'IE' Interrupt enable.
 'IM' Interrupt mask.
 'IP' Interrupt pending.
 'ICC' Instruction cache control.
 'DCC' Data cache control.
 'CC' Cycle counter.
 'CFG' Configuration.
 'EBA' Exception base address.
 'DC' Debug control.
 'DEBA' Debug exception base address.
 'JTX' JTAG transmit.
 'JRX' JTAG receive.
 'BP0'

```

Breakpoint 0.
'BP1'
Breakpoint 1.
'BP2'
Breakpoint 2.
'BP3'
Breakpoint 3.
'WP0'
Watchpoint 0.
'WP1'
Watchpoint 1.
'WP2'
Watchpoint 2.
'WP3'
Watchpoint 3.

```



File: as.info, Node: LM32-Modifiers, Next: LM32-Chars, Prev: LM32-Regs, Up: LM32 Syntax

9.20.2.2 Relocatable Expression Modifiers

.....

The assembler supports several modifiers when using relocatable addresses in LM32 instruction operands. The general syntax is the following:

```
modifier(relocatable-expression)
```

'lo'

This modifier allows you to use bits 0 through 15 of an address expression as 16 bit relocatable expression.

'hi'

This modifier allows you to use bits 16 through 23 of an address expression as 16 bit relocatable expression.

For example

```
ori r4, r4, lo(sym+10)
orhi r4, r4, hi(sym+10)
```

'gp'

This modified creates a 16-bit relocatable expression that is the offset of the symbol from the global pointer.

```
mva r4, gp(sym)
```

'got'

This modifier places a symbol in the GOT and creates a 16-bit relocatable expression that is the offset into the GOT of this symbol.

```
lw r4, (gp+got(sym))
```

'gotofflo16'

This modifier allows you to use the bits 0 through 15 of an address which is an offset from the GOT.

'gotoffhi16'

This modifier allows you to use the bits 16 through 31 of an address which is an offset from the GOT.

```
orhi r4, r4, gotoffhi16(lsym)
addi r4, r4, gotofflo16(lsym)
```



File: as.info, Node: LM32-Chars, Prev: LM32-Modifiers, Up: LM32 Syntax

9.20.2.3 Special Characters

.....

The presence of a '#' on a line indicates the start of a comment that extends to the end of the current line. Note that if a line starts with a '#' character then it can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

A semicolon(';') can be used to separate multiple statements on the same line.



File: as.info, Node: LM32 Opcodes, Prev: LM32 Syntax, Up: LM32-Dependent

9.20.3 Opcodes

For detailed information on the LM32 machine instruction set, see <<http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/>>.

'as' implements all the standard LM32 opcodes.



File: as.info, Node: M32C-Dependent, Next: M32R-Dependent, Prev: LM32-Dependent, Up: Machine Dependencies

9.21 M32C Dependent Features

=====

'as' can assemble code for several different members of the Renesas M32C family. Normally the default is to assemble code for the M16C microprocessor. The '-m32c' option may be used to change the default to the M32C microprocessor.

* Menu:

```
* M32C-Opts::          M32C Options
* M32C-Syntax::      M32C Syntax
```



File: as.info, Node: M32C-Opts, Next: M32C-Syntax, Up: M32C-Dependent

9.21.1 M32C Options

The Renesas M32C version of 'as' has these machine-dependent options:

- '-m32c'
Assemble M32C instructions.
- '-m16c'
Assemble M16C instructions (default).
- '-relax'
Enable support for link-time relaxations.
- '-h-tick-hex'
Support H'00 style hex constants in addition to 0x00 style.



File: as.info, Node: M32C-Syntax, Prev: M32C-Opts, Up: M32C-Dependent

9.21.2 M32C Syntax

* Menu:

- * M32C-Modifiers:: Symbolic Operand Modifiers
- * M32C-Chars:: Special Characters



File: as.info, Node: M32C-Modifiers, Next: M32C-Chars, Up: M32C-Syntax

9.21.2.1 Symbolic Operand Modifiers

.....

The assembler supports several modifiers when using symbol addresses in M32C instruction operands. The general syntax is the following:

```
%modifier(symbol)
```

```
'%dsp8'  
'%dsp16'
```

These modifiers override the assembler's assumptions about how big a symbol's address is. Normally, when it sees an operand like 'sym[a0]' it assumes 'sym' may require the widest displacement field (16 bits for '-m16c', 24 bits for '-m32c'). These modifiers tell it to assume the address will fit in an 8 or 16 bit (respectively) unsigned displacement. Note that, of course, if it doesn't actually fit you will get linker errors. Example:

```
mov.w %dsp8(sym)[a0],r1  
mov.b #0,%dsp8(sym)[a0]
```

```
'%hi8'
```

This modifier allows you to load bits 16 through 23 of a 24 bit address into an 8 bit register. This is useful with, for example, the M16C 'smovf' instruction, which expects a 20 bit address in 'r1h' and 'a0'. Example:

```

mov.b #%hi8(sym),r1h
mov.w #%lo16(sym),a0
smovf.b

```

'%lo16'

Likewise, this modifier allows you to load bits 0 through 15 of a 24 bit address into a 16 bit register.

'%hi16'

This modifier allows you to load bits 16 through 31 of a 32 bit address into a 16 bit register. While the M32C family only has 24 bits of address space, it does support addresses in pairs of 16 bit registers (like 'a1a0' for the 'lde' instruction). This modifier is for loading the upper half in such cases. Example:

```

mov.w #%hi16(sym),a1
mov.w #%lo16(sym),a0
...
lde.w [a1a0],r1

```



File: as.info, Node: M32C-Chars, Prev: M32C-Modifiers, Up: M32C-Syntax

9.21.2.2 Special Characters

.....

The presence of a ';' character on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The '|' character can be used to separate statements on the same line.



File: as.info, Node: M32R-Dependent, Next: M68K-Dependent, Prev: M32C-Dependent, Up: Machine Dependencies

9.22 M32R Dependent Features

=====

* Menu:

```

* M32R-Opts::                M32R Options
* M32R-Directives::         M32R Directives
* M32R-Warnings::          M32R Warnings

```



File: as.info, Node: M32R-Opts, Next: M32R-Directives, Up: M32R-Dependent

9.22.1 M32R Options

The Release M32R version of 'as' has a few machine dependent options:

'-m32rx'

'as' can assemble code for several different members of the Renesas M32R family. Normally the default is to assemble code for the M32R microprocessor. This option may be used to change the default to the M32RX microprocessor, which adds some more instructions to the basic M32R instruction set, and some additional parameters to some of the original instructions.

'-m32r2'

This option changes the target processor to the M32R2 microprocessor.

'-m32r'

This option can be used to restore the assembler's default behaviour of assembling for the M32R microprocessor. This can be useful if the default has been changed by a previous command line option.

'-little'

This option tells the assembler to produce little-endian code and data. The default is dependent upon how the toolchain was configured.

'-EL'

This is a synonym for `_-little_`.

'-big'

This option tells the assembler to produce big-endian code and data.

'-EB'

This is a synonym for `_-big_`.

'-KPIC'

This option specifies that the output of the assembler should be marked as position-independent code (PIC).

'-parallel'

This option tells the assembler to attempt to combine two sequential instructions into a single, parallel instruction, where it is legal to do so.

'-no-parallel'

This option disables a previously enabled `_-parallel_` option.

'-no-bitinst'

This option disables the support for the extended bit-field instructions provided by the M32R2. If this support needs to be re-enabled the `_-bitinst_` switch can be used to restore it.

'-O'

This option tells the assembler to attempt to optimize the instructions that it produces. This includes filling delay slots and converting sequential instructions into parallel ones. This option implies `_-parallel_`.

'-warn-explicit-parallel-conflicts'

Instructs 'as' to produce warning messages when questionable parallel instructions are encountered. This option is enabled by default, but 'gcc' disables it when it invokes 'as' directly. Questionable instructions are those whose behaviour would be different if they were executed sequentially. For example the code fragment 'mv r1, r2 || mv r3, r1' produces a different result from 'mv r1, r2 \n mv r3, r1' since the former moves r1 into r3 and then r2 into r1, whereas the later moves r2 into r1 and r3.

'-Wp'

This is a shorter synonym for the `__warn-explicit-parallel-conflicts_` option.

'-no-warn-explicit-parallel-conflicts'

Instructs 'as' not to produce warning messages when questionable parallel instructions are encountered.

'-Wnp'

This is a shorter synonym for the `__no-warn-explicit-parallel-conflicts_` option.

'-ignore-parallel-conflicts'

This option tells the assembler's to stop checking parallel instructions for constraint violations. This ability is provided for hardware vendors testing chip designs and should not be used under normal circumstances.

'-no-ignore-parallel-conflicts'

This option restores the assembler's default behaviour of checking parallel instructions to detect constraint violations.

'-Ip'

This is a shorter synonym for the `__ignore-parallel-conflicts_` option.

'-nIp'

This is a shorter synonym for the `__no-ignore-parallel-conflicts_` option.

'-warn-unmatched-high'

This option tells the assembler to produce a warning message if a '.high' pseudo op is encountered without a matching '.low' pseudo op. The presence of such an unmatched pseudo op usually indicates a programming error.

'-no-warn-unmatched-high'

Disables a previously enabled `__warn-unmatched-high_` option.

'-Wuh'

This is a shorter synonym for the `__warn-unmatched-high_` option.

'-Wnuh'

This is a shorter synonym for the `__no-warn-unmatched-high_` option.



 The Renease M32R version of 'as' has a few architecture specific directives:

'low EXPRESSION'

The 'low' directive computes the value of its expression and places the lower 16-bits of the result into the immediate-field of the instruction. For example:

```
or3   r0, r0, #low(0x12345678) ; compute r0 = r0 | 0x5678
add3, r0, r0, #low(fred)      ; compute r0 = r0 + low 16-bits of address of
                                fred
```

'high EXPRESSION'

The 'high' directive computes the value of its expression and places the upper 16-bits of the result into the immediate-field of the instruction. For example:

```
seth  r0, #high(0x12345678) ; compute r0 = 0x12340000
seth, r0, #high(fred)      ; compute r0 = upper 16-bits of address of
                                fred
```

'shigh EXPRESSION'

The 'shigh' directive is very similar to the 'high' directive. It also computes the value of its expression and places the upper 16-bits of the result into the immediate-field of the instruction. The difference is that 'shigh' also checks to see if the lower 16-bits could be interpreted as a signed number, and if so it assumes that a borrow will occur from the upper-16 bits. To compensate for this the 'shigh' directive pre-biases the upper 16 bit value by adding one to it. For example:

For example:

```
seth  r0, #shigh(0x12345678) ; compute r0 = 0x12340000
seth  r0, #shigh(0x00008000) ; compute r0 = 0x00010000
```

In the second example the lower 16-bits are 0x8000. If these are treated as a signed value and sign extended to 32-bits then the value becomes 0xffff8000. If this value is then added to 0x00010000 then the result is 0x00008000.

This behaviour is to allow for the different semantics of the 'or3' and 'add3' instructions. The 'or3' instruction treats its 16-bit immediate argument as unsigned whereas the 'add3' treats its 16-bit immediate as a signed value. So for example:

```
seth  r0, #shigh(0x00008000)
add3  r0, r0, #low(0x00008000)
```

Produces the correct result in r0, whereas:

```
seth  r0, #shigh(0x00008000)
or3   r0, r0, #low(0x00008000)
```

Stores 0xffff8000 into r0.

Note - the 'shigh' directive does not know where in the assembly

source code the lower 16-bits of the value are going set, so it cannot check to make sure that an 'or3' instruction is being used rather than an 'add3' instruction. It is up to the programmer to make sure that correct directives are used.

`'.m32r'`

The directive performs a similar thing as the `__m32r_` command line option. It tells the assembler to only accept M32R instructions from now on. An instructions from later M32R architectures are refused.

`'.m32rx'`

The directive performs a similar thing as the `__m32rx_` command line option. It tells the assembler to start accepting the extra instructions in the M32RX ISA as well as the ordinary M32R ISA.

`'.m32r2'`

The directive performs a similar thing as the `__m32r2_` command line option. It tells the assembler to start accepting the extra instructions in the M32R2 ISA as well as the ordinary M32R ISA.

`'.little'`

The directive performs a similar thing as the `__little_` command line option. It tells the assembler to start producing little-endian code and data. This option should be used with care as producing mixed-endian binary files is fraught with danger.

`'.big'`

The directive performs a similar thing as the `__big_` command line option. It tells the assembler to start producing big-endian code and data. This option should be used with care as producing mixed-endian binary files is fraught with danger.



File: as.info, Node: M32R-Warnings, Prev: M32R-Directives, Up: M32R-Dependent

9.22.3 M32R Warnings

There are several warning and error messages that can be produced by 'as' which are specific to the M32R:

'output of 1st instruction is the same as an input to 2nd instruction - is this intentional ?'

This message is only produced if warnings for explicit parallel conflicts have been enabled. It indicates that the assembler has encountered a parallel instruction in which the destination register of the left hand instruction is used as an input register in the right hand instruction. For example in this code fragment 'mv r1, r2 || neg r3, r1' register r1 is the destination of the move instruction and the input to the neg instruction.

'output of 2nd instruction is the same as an input to 1st instruction - is this intentional ?'

This message is only produced if warnings for explicit parallel conflicts have been enabled. It indicates that the assembler has encountered a parallel instruction in which the destination register of the right hand instruction is used as an input register in the left hand instruction. For example in this code fragment

'mv r1, r2 || neg r2, r3' register r2 is the destination of the neg instruction and the input to the move instruction.

'instruction '...' is for the M32RX only'

This message is produced when the assembler encounters an instruction which is only supported by the M32Rx processor, and the '-m32rx' command line flag has not been specified to allow assembly of such instructions.

'unknown instruction '...''

This message is produced when the assembler encounters an instruction which it does not recognize.

'only the NOP instruction can be issued in parallel on the m32r'

This message is produced when the assembler encounters a parallel instruction which does not involve a NOP instruction and the '-m32rx' command line flag has not been specified. Only the M32Rx processor is able to execute two instructions in parallel.

'instruction '...' cannot be executed in parallel.'

This message is produced when the assembler encounters a parallel instruction which is made up of one or two instructions which cannot be executed in parallel.

'Instructions share the same execution pipeline'

This message is produced when the assembler encounters a parallel instruction whose components both use the same execution pipeline.

'Instructions write to the same destination register.'

This message is produced when the assembler encounters a parallel instruction where both components attempt to modify the same register. For example these code fragments will produce this message: 'mv r1, r2 || neg r1, r3' 'j1 r0 || mv r14, r1' 'st r2, @-r1 || mv r1, r3' 'mv r1, r2 || ld r0, @r1+' 'cmp r1, r2 || addx r3, r4' (Both write to the condition bit)



File: as.info, Node: M68K-Dependent, Next: M68HC11-Dependent, Prev: M32R-Dependent, Up: Machine Dependencies

9.23 M680x0 Dependent Features

=====

* Menu:

* M68K-Opts::	M680x0 Options
* M68K-Syntax::	Syntax
* M68K-Moto-Syntax::	Motorola Syntax
* M68K-Float::	Floating Point
* M68K-Directives::	680x0 Machine Directives
* M68K-opcodes::	Opcodes



File: as.info, Node: M68K-Opts, Next: M68K-Syntax, Up: M68K-Dependent

9.23.1 M680x0 Options

The Motorola 680x0 version of 'as' has a few machine dependent options:

'-march=ARCHITECTURE'

This option specifies a target architecture. The following architectures are recognized: '68000', '68010', '68020', '68030', '68040', '68060', 'cpu32', 'isaa', 'isaaplus', 'isab', 'isac' and 'cfv4e'.

'-mcpu=CPU'

This option specifies a target cpu. When used in conjunction with the '-march' option, the cpu must be within the specified architecture. Also, the generic features of the architecture are used for instruction generation, rather than those of the specific chip.

'-m[no-]68851'

'-m[no-]68881'

'-m[no-]div'

'-m[no-]usp'

'-m[no-]float'

'-m[no-]mac'

'-m[no-]emac'

Enable or disable various architecture specific features. If a chip or architecture by default supports an option (for instance '-march=isaaplus' includes the '-mdiv' option), explicitly disabling the option will override the default.

'-l'

You can use the '-l' option to shorten the size of references to undefined symbols. If you do not use the '-l' option, references to undefined symbols are wide enough for a full 'long' (32 bits). (Since 'as' cannot know where these symbols end up, 'as' can only allocate space for the linker to fill in later. Since 'as' does not know how far away these symbols are, it allocates as much space as it can.) If you use this option, the references are only one word wide (16 bits). This may be useful if you want the object file to be as small as possible, and you know that the relevant symbols are always less than 17 bits away.

'--register-prefix-optional'

For some configurations, especially those where the compiler normally does not prepend an underscore to the names of user variables, the assembler requires a '%' before any use of a register name. This is intended to let the assembler distinguish between C variables and functions named 'a0' through 'a7', and so on. The '%' is always accepted, but is not required for certain configurations, notably 'sun3'. The '--register-prefix-optional' option may be used to permit omitting the '%' even for configurations for which it is normally required. If this is done, it will generally be impossible to refer to C variables and functions with the same names as register names.

'--bitwise-or'

Normally the character '|' is treated as a comment character, which means that it can not be used in expressions. The '--bitwise-or' option turns '|' into a normal character. In this mode, you must either use C style comments, or start comments with a '#' character at the beginning of a line.

'--base-size-default-16 --base-size-default-32'

If you use an addressing mode with a base register without specifying the size, 'as' will normally use the full 32 bit value. For example, the addressing mode '%a0@(%d0)' is equivalent to '%a0@(%d0:l)'. You may use the '--base-size-default-16' option to tell 'as' to default to using the 16 bit value. In this case, '%a0@(%d0)' is equivalent to '%a0@(%d0:w)'. You may use the '--base-size-default-32' option to restore the default behaviour.

'--disp-size-default-16 --disp-size-default-32'

If you use an addressing mode with a displacement, and the value of the displacement is not known, 'as' will normally assume that the value is 32 bits. For example, if the symbol 'disp' has not been defined, 'as' will assemble the addressing mode '%a0@(disp,%d0)' as though 'disp' is a 32 bit value. You may use the '--disp-size-default-16' option to tell 'as' to instead assume that the displacement is 16 bits. In this case, 'as' will assemble '%a0@(disp,%d0)' as though 'disp' is a 16 bit value. You may use the '--disp-size-default-32' option to restore the default behaviour.

'--pcrel'

Always keep branches PC-relative. In the M680x0 architecture all branches are defined as PC-relative. However, on some processors they are limited to word displacements maximum. When 'as' needs a long branch that is not available, it normally emits an absolute jump instead. This option disables this substitution. When this option is given and no long branches are available, only word branches will be emitted. An error message will be generated if a word branch cannot reach its target. This option has no effect on 68020 and other processors that have long branches. *note Branch Improvement: M68K-Branch.

'-m68000'

'as' can assemble code for several different members of the Motorola 680x0 family. The default depends upon how 'as' was configured when it was built; normally, the default is to assemble code for the 68020 microprocessor. The following options may be used to change the default. These options control which instructions and addressing modes are permitted. The members of the 680x0 family are very similar. For detailed information about the differences, see the Motorola manuals.

'-m68000'

'-m68ec000'

'-m68hc000'

'-m68hc001'

'-m68008'

'-m68302'

'-m68306'

'-m68307'

'-m68322'

'-m68356'

Assemble for the 68000. '-m68008', '-m68302', and so on are synonyms for '-m68000', since the chips are the same from the point of view of the assembler.

'-m68010'

Assemble for the 68010.

'-m68020'
'-m68ec020'
Assemble for the 68020. This is normally the default.

'-m68030'
'-m68ec030'
Assemble for the 68030.

'-m68040'
'-m68ec040'
Assemble for the 68040.

'-m68060'
'-m68ec060'
Assemble for the 68060.

'-mcpu32'
'-m68330'
'-m68331'
'-m68332'
'-m68333'
'-m68334'
'-m68336'
'-m68340'
'-m68341'
'-m68349'
'-m68360'
Assemble for the CPU32 family of chips.

'-m5200'
'-m5202'
'-m5204'
'-m5206'
'-m5206e'
'-m521x'
'-m5249'
'-m528x'
'-m5307'
'-m5407'
'-m547x'
'-m548x'
'-mcfv4'
'-mcfv4e'
Assemble for the ColdFire family of chips.

'-m68881'
'-m68882'
Assemble 68881 floating point instructions. This is the default for the 68020, 68030, and the CPU32. The 68040 and 68060 always support floating point instructions.

'-mno-68881'
Do not assemble 68881 floating point instructions. This is the default for 68000 and the 68010. The 68040 and 68060 always support floating point instructions, even if this option is used.

'-m68851'

Assemble 68851 MMU instructions. This is the default for the 68020, 68030, and 68060. The 68040 accepts a somewhat different set of MMU instructions; '-m68851' and '-m68040' should not be used together.

'-mno-68851'

Do not assemble 68851 MMU instructions. This is the default for the 68000, 68010, and the CPU32. The 68040 accepts a somewhat different set of MMU instructions.



File: as.info, Node: M68K-Syntax, Next: M68K-Moto-Syntax, Prev: M68K-Opts, Up: M68K-Dependent

9.23.2 Syntax

This syntax for the Motorola 680x0 was developed at MIT.

The 680x0 version of 'as' uses instructions names and syntax compatible with the Sun assembler. Intervening periods are ignored; for example, 'movl' is equivalent to 'mov.l'.

In the following table APC stands for any of the address registers ('%a0' through '%a7'), the program counter ('%pc'), the zero-address relative to the program counter ('%zpc'), a suppressed address register ('%za0' through '%za7'), or it may be omitted entirely. The use of SIZE means one of 'w' or 'l', and it may be omitted, along with the leading colon, unless a scale is also specified. The use of SCALE means one of '1', '2', '4', or '8', and it may always be omitted along with the leading colon.

The following addressing modes are understood:

"Immediate"

'#NUMBER'

"Data Register"

'%d0' through '%d7'

"Address Register"

'%a0' through '%a7'

'%a7' is also known as '%sp', i.e., the Stack Pointer. '%a6' is also known as '%fp', the Frame Pointer.

"Address Register Indirect"

'%a0@' through '%a7@'

"Address Register Postincrement"

'%a0@+' through '%a7@+'

"Address Register Predecrement"

'%a0@-' through '%a7@-'

"Indirect Plus Offset"

'APC@(NUMBER)'

"Index"

'APC@(NUMBER,REGISTER:SIZE:SCALE)'

The NUMBER may be omitted.

"Postindex"

'APC@(NUMBER)@(ONUMBER,REGISTER:SIZE:SCALE)'

The ONUMBER or the REGISTER, but not both, may be omitted.

"Preindex"

'APC@(NUMBER,REGISTER:SIZE:SCALE)@(ONUMBER)'

The NUMBER may be omitted. Omitting the REGISTER produces the Postindex addressing mode.

"Absolute"

'SYMBOL', or 'DIGITS', optionally followed by ':b', ':w', or ':l'.



File: as.info, Node: M68K-Moto-Syntax, Next: M68K-Float, Prev: M68K-Syntax, Up: M68K-Dependent

9.23.3 Motorola Syntax

The standard Motorola syntax for this chip differs from the syntax already discussed (*note Syntax: M68K-Syntax.). 'as' can accept Motorola syntax for operands, even if MIT syntax is used for other operands in the same instruction. The two kinds of syntax are fully compatible.

In the following table APC stands for any of the address registers ('%a0' through '%a7'), the program counter ('%pc'), the zero-address relative to the program counter ('%zpc'), or a suppressed address register ('%za0' through '%za7'). The use of SIZE means one of 'w' or 'l', and it may always be omitted along with the leading dot. The use of SCALE means one of '1', '2', '4', or '8', and it may always be omitted along with the leading asterisk.

The following additional addressing modes are understood:

"Address Register Indirect"

'(%a0)' through '%a7)'

'%a7' is also known as '%sp', i.e., the Stack Pointer. '%a6' is also known as '%fp', the Frame Pointer.

"Address Register Postincrement"

'(%a0)+' through '%a7)+'

"Address Register Predecrement"

'-(%a0)' through '-(%a7)'

"Indirect Plus Offset"

'NUMBER(%A0)' through 'NUMBER(%A7)', or 'NUMBER(%PC)'.

The NUMBER may also appear within the parentheses, as in '(NUMBER,%A0)'. When used with the PC, the NUMBER may be omitted (with an address register, omitting the NUMBER produces Address Register Indirect mode).

"Index"

```
'NUMBER(APC,REGISTER.SIZE*SCALE)'
```

The NUMBER may be omitted, or it may appear within the parentheses. The APC may be omitted. The REGISTER and the APC may appear in either order. If both APC and REGISTER are address registers, and the SIZE and SCALE are omitted, then the first register is taken as the base register, and the second as the index register.

"Postindex"

```
'([NUMBER,APC],REGISTER.SIZE*SCALE,ONUMBER)'
```

The ONUMBER, or the REGISTER, or both, may be omitted. Either the NUMBER or the APC may be omitted, but not both.

"Preindex"

```
'([NUMBER,APC,REGISTER.SIZE*SCALE],ONUMBER)'
```

The NUMBER, or the APC, or the REGISTER, or any two of them, may be omitted. The ONUMBER may be omitted. The REGISTER and the APC may appear in either order. If both APC and REGISTER are address registers, and the SIZE and SCALE are omitted, then the first register is taken as the base register, and the second as the index register.

US

File: as.info, Node: M68K-Float, Next: M68K-Directives, Prev: M68K-Moto-Syntax, Up: M68K-Dependent

9.23.4 Floating Point

Packed decimal (P) format floating literals are not supported. Feel free to add the code!

The floating point formats generated by directives are these.

'float'

'Single' precision floating point constants.

'double'

'Double' precision floating point constants.

'extend'

'ldouble'

'Extended' precision ('long double') floating point constants.

US

File: as.info, Node: M68K-Directives, Next: M68K-opcodes, Prev: M68K-Float, Up: M68K-Dependent

9.23.5 680x0 Machine Directives

In order to be compatible with the Sun assembler the 680x0 assembler understands the following directives.

'data1'

This directive is identical to a '.data 1' directive.

`'.data2'`

This directive is identical to a `'.data 2'` directive.

`'.even'`

This directive is a special case of the `'.align'` directive; it aligns the output to an even byte boundary.

`'.skip'`

This directive is identical to a `'.space'` directive.

`'.arch NAME'`

Select the target architecture and extension features. Valid values for NAME are the same as for the `'-march'` command line option. This directive cannot be specified after any instructions have been assembled. If it is given multiple times, or in conjunction with the `'-march'` option, all uses must be for the same architecture and extension set.

`'.cpu NAME'`

Select the target cpu. Valid value for NAME are the same as for the `'-mcpu'` command line option. This directive cannot be specified after any instructions have been assembled. If it is given multiple times, or in conjunction with the `'-mopt'` option, all uses must be for the same cpu.



File: as.info, Node: M68K-opcodes, Prev: M68K-Directives, Up: M68K-Dependent

9.23.6 Opcodes

* Menu:

* M68K-Branch:: Branch Improvement
* M68K-Chars:: Special Characters



File: as.info, Node: M68K-Branch, Next: M68K-Chars, Up: M68K-opcodes

9.23.6.1 Branch Improvement

.....

Certain pseudo opcodes are permitted for branch instructions. They expand to the shortest branch instruction that reach the target. Generally these mnemonics are made by substituting 'j' for 'b' at the start of a Motorola mnemonic.

The following table summarizes the pseudo-operations. A '*' flags cases that are more fully described after the table:

Pseudo-Op	Displacement			
	BYTE	WORD	LONG	68000/10, not PC-relative OK
jbsr	bsrs	bsrw	bsrl	jsr
jra	bras	braw	bral	jmp
* jXX	bXXs	bXXw	bXXl	bNXs;jmp
* dbXX	N/A	dbXXw	dbXX;bras;bral	dbXX;bras;jmp

fjXX | N/A fbXXw fbXXl N/A

XX: condition

NX: negative of condition XX

'*''--see full description below

'**''--this expansion mode is disallowed by '--pcrel'

'jbsr'

'jra'

These are the simplest jump pseudo-operations; they always map to one particular machine instruction, depending on the displacement to the branch target. This instruction will be a byte or word branch is that is sufficient. Otherwise, a long branch will be emitted if available. If no long branches are available and the '--pcrel' option is not given, an absolute long jump will be emitted instead. If no long branches are available, the '--pcrel' option is given, and a word branch cannot reach the target, an error message is generated.

In addition to standard branch operands, 'as' allows these pseudo-operations to have all operands that are allowed for jsr and jmp, substituting these instructions if the operand given is not valid for a branch instruction.

'jXX'

Here, 'jXX' stands for an entire family of pseudo-operations, where XX is a conditional branch or condition-code test. The full list of pseudo-ops in this family is:

```

    jhi   jls   jcc   jcs   jne   jeq   jvc
    jvs   jpl   jmi   jge   jlt   jgt   jle

```

Usually, each of these pseudo-operations expands to a single branch instruction. However, if a word branch is not sufficient, no long branches are available, and the '--pcrel' option is not given, 'as' issues a longer code fragment in terms of NX, the opposite condition to XX. For example, under these conditions:

```

    jXX foo
gives
    bNXs oof
    jmp foo
oof:

```

'dbXX'

The full family of pseudo-operations covered here is

```

    dbhi   dbls   dbcc   dbcs   dbne   dbeq   dbvc
    dbvs   dbpl   dbmi   dbge   dblt   dbgt   dble
    dbf    dbra   dbt

```

Motorola 'dbXX' instructions allow word displacements only. When a word displacement is sufficient, each of these pseudo-operations expands to the corresponding Motorola instruction. When a word displacement is not sufficient and long branches are available, when the source reads 'dbXX foo', 'as' emits

```

    dbXX oo1
    bras oo2
oo1:bral foo
oo2:

```

If, however, long branches are not available and the '--pcrel' option is not given, 'as' emits

```
    dbXX oo1
    bras oo2
oo1: jmp foo
oo2:
```

'fjXX'

This family includes

```
    fjne  fjeq  fjge  fjlt  fjgt  fjle  fjf
    fjt   fjgl  fjgle fjnge fjngl fjngle fjngt
    fjnle fjnlt fjoge fjogl fjogt fjole fjolt
    fjor  fjseq fjsf  fjsne fjst  fjueq fjuge
    fjugt fjule fjult fjun
```

Each of these pseudo-operations always expands to a single Motorola coprocessor branch instruction, word or long. All Motorola coprocessor branch instructions allow both word and long displacements.



File: as.info, Node: M68K-Chars, Prev: M68K-Branch, Up: M68K-opcodes

9.23.6.2 Special Characters

.....

Line comments are introduced by the '|' character appearing anywhere on a line, unless the '--bitwise-or' command line option has been specified.

An asterisk ('*') as the first character on a line marks the start of a line comment as well.

A hash character ('#') as the first character on a line also marks the start of a line comment, but in this case it could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:). If the hash character appears elsewhere on a line it is used to introduce an immediate value. (This is for compatibility with Sun's assembler).

Multiple statements on the same line can appear if they are separated by the ';' character.



File: as.info, Node: M68HC11-Dependent, Next: Meta-Dependent, Prev: M68K-Dependent, Up: Machine Dependencies

9.24 M68HC11 and M68HC12 Dependent Features

=====

* Menu:

```
* M68HC11-Opts::          M68HC11 and M68HC12 Options
* M68HC11-Syntax::      Syntax
* M68HC11-Modifiers::    Symbolic Operand Modifiers
* M68HC11-Directives::  Assembler Directives
* M68HC11-Float::       Floating Point
* M68HC11-opcodes::     Opcodes
```



File: as.info, Node: M68HC11-Opts, Next: M68HC11-Syntax, Up: M68HC11-Dependent

9.24.1 M68HC11 and M68HC12 Options

The Motorola 68HC11 and 68HC12 version of 'as' have a few machine dependent options.

'-m68hc11'

This option switches the assembler into the M68HC11 mode. In this mode, the assembler only accepts 68HC11 operands and mnemonics. It produces code for the 68HC11.

'-m68hc12'

This option switches the assembler into the M68HC12 mode. In this mode, the assembler also accepts 68HC12 operands and mnemonics. It produces code for the 68HC12. A few 68HC11 instructions are replaced by some 68HC12 instructions as recommended by Motorola specifications.

'-m68hcs12'

This option switches the assembler into the M68HCS12 mode. This mode is similar to '-m68hc12' but specifies to assemble for the 68HCS12 series. The only difference is on the assembling of the 'movb' and 'movw' instruction when a PC-relative operand is used.

'-mm9s12x'

This option switches the assembler into the M9S12X mode. This mode is similar to '-m68hc12' but specifies to assemble for the S12X series which is a superset of the HCS12.

'-mm9s12xg'

This option switches the assembler into the XGATE mode for the RISC co-processor featured on some S12X-family chips.

'--xgate-ramoffset'

This option instructs the linker to offset RAM addresses from S12X address space into XGATE address space.

'-mshort'

This option controls the ABI and indicates to use a 16-bit integer ABI. It has no effect on the assembled instructions. This is the default.

'-mlong'

This option controls the ABI and indicates to use a 32-bit integer ABI.

'-mshort-double'

This option controls the ABI and indicates to use a 32-bit float ABI. This is the default.

'-mlong-double'

This option controls the ABI and indicates to use a 64-bit float ABI.

'--strict-direct-mode'

You can use the '--strict-direct-mode' option to disable the

automatic translation of direct page mode addressing into extended mode when the instruction does not support direct mode. For example, the 'clr' instruction does not support direct page mode addressing. When it is used with the direct page mode, 'as' will ignore it and generate an absolute addressing. This option prevents 'as' from doing this, and the wrong usage of the direct page mode will raise an error.

'--short-branches'

The '--short-branches' option turns off the translation of relative branches into absolute branches when the branch offset is out of range. By default 'as' transforms the relative branch ('bsr', 'bgt', 'bge', 'beq', 'bne', 'ble', 'blt', 'bhi', 'bcc', 'bls', 'bcs', 'bmi', 'bvs', 'bvs', 'bra') into an absolute branch when the offset is out of the -128 .. 127 range. In that case, the 'bsr' instruction is translated into a 'jsr', the 'bra' instruction is translated into a 'jmp' and the conditional branches instructions are inverted and followed by a 'jmp'. This option disables these translations and 'as' will generate an error if a relative branch is out of range. This option does not affect the optimization associated to the 'jbra', 'jbsr' and 'jbXX' pseudo opcodes.

'--force-long-branches'

The '--force-long-branches' option forces the translation of relative branches into absolute branches. This option does not affect the optimization associated to the 'jbra', 'jbsr' and 'jbXX' pseudo opcodes.

'--print-insn-syntax'

You can use the '--print-insn-syntax' option to obtain the syntax description of the instruction when an error is detected.

'--print-opcodes'

The '--print-opcodes' option prints the list of all the instructions with their syntax. The first item of each line represents the instruction name and the rest of the line indicates the possible operands for that instruction. The list is printed in alphabetical order. Once the list is printed 'as' exits.

'--generate-example'

The '--generate-example' option is similar to '--print-opcodes' but it generates an example for each instruction instead.



File: as.info, Node: M68HC11-Syntax, Next: M68HC11-Modifiers, Prev: M68HC11-Opts, Up: M68HC11-Dependent

9.24.2 Syntax

In the M68HC11 syntax, the instruction name comes first and it may be followed by one or several operands (up to three). Operands are separated by comma (','), In the normal mode, 'as' will complain if too many operands are specified for a given instruction. In the MRI mode (turned on with '-M' option), it will treat them as comments. Example:

```
inx
lda #23
bset 2,x #4
```



```
brclr *bot #8 foo
```

The presence of a ';' character or a '!' character anywhere on a line indicates the start of a comment that extends to the end of that line.

A '*' or a '#' character at the start of a line also introduces a line comment, but these characters do not work elsewhere on the line. If the first character of the line is a '#' then as well as starting a comment, the line could also be logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The M68HC11 assembler does not currently support a line separator character.

The following addressing modes are understood for 68HC11 and 68HC12: "Immediate"

```
'#NUMBER'
```

"Address Register"

```
'NUMBER,X', 'NUMBER,Y'
```

The NUMBER may be omitted in which case 0 is assumed.

"Direct Addressing mode"

```
'*SYMBOL', or '*DIGITS'
```

"Absolute"

```
'SYMBOL', or 'DIGITS'
```

The M68HC12 has other more complex addressing modes. All of them are supported and they are represented below:

"Constant Offset Indexed Addressing Mode"

```
'NUMBER,REG'
```

The NUMBER may be omitted in which case 0 is assumed. The register can be either 'X', 'Y', 'SP' or 'PC'. The assembler will use the smaller post-byte definition according to the constant value (5-bit constant offset, 9-bit constant offset or 16-bit constant offset). If the constant is not known by the assembler it will use the 16-bit constant offset post-byte and the value will be resolved at link time.

"Offset Indexed Indirect"

```
'[NUMBER,REG]'
```

The register can be either 'X', 'Y', 'SP' or 'PC'.

"Auto Pre-Increment/Pre-Decrement/Post-Increment/Post-Decrement"

```
'NUMBER,-REG' 'NUMBER,+REG' 'NUMBER,REG-' 'NUMBER,REG+'
```

The number must be in the range '-8'..' +8' and must not be 0. The register can be either 'X', 'Y', 'SP' or 'PC'.

"Accumulator Offset"

```
'ACC,REG'
```

The accumulator register can be either 'A', 'B' or 'D'. The register can be either 'X', 'Y', 'SP' or 'PC'.

```
"Accumulator D offset indexed-indirect"
  '[D,REG]'
```

The register can be either 'X', 'Y', 'SP' or 'PC'.

For example:

```
ldab 1024,sp
ldd [10,x]
orab 3,+x
stab -2,y-
ldx a,pc
sty [d,sp]
```



File: as.info, Node: M68HC11-Modifiers, Next: M68HC11-Directives, Prev: M68HC11-Syntax, Up: M68HC11-Dependent

9.24.3 Symbolic Operand Modifiers

The assembler supports several modifiers when using symbol addresses in 68HC11 and 68HC12 instruction operands. The general syntax is the following:

```
%modifier(symbol)
```

'%addr'

This modifier indicates to the assembler and linker to use the 16-bit physical address corresponding to the symbol. This is intended to be used on memory window systems to map a symbol in the memory bank window. If the symbol is in a memory expansion part, the physical address corresponds to the symbol address within the memory bank window. If the symbol is not in a memory expansion part, this is the symbol address (using or not using the %addr modifier has no effect in that case).

'%page'

This modifier indicates to use the memory page number corresponding to the symbol. If the symbol is in a memory expansion part, its page number is computed by the linker as a number used to map the page containing the symbol in the memory bank window. If the symbol is not in a memory expansion part, the page number is 0.

'%hi'

This modifier indicates to use the 8-bit high part of the physical address of the symbol.

'%lo'

This modifier indicates to use the 8-bit low part of the physical address of the symbol.

For example a 68HC12 call to a function 'foo_example' stored in memory expansion part could be written as follows:

```
call %addr(foo_example),%page(foo_example)
```

and this is equivalent to

```
call foo_example
```

And for 68HC11 it could be written as follows:

```
ldab #%page(foo_example)
stab _page_switch
jsr %addr(foo_example)
```

US

File: as.info, Node: M68HC11-Directives, Next: M68HC11-Float, Prev: M68HC11-Modifiers, Up: M68HC11-Dependent

9.24.4 Assembler Directives

The 68HC11 and 68HC12 version of 'as' have the following specific assembler directives:

'`.relax`'

The `relax` directive is used by the 'GNU Compiler' to emit a specific relocation to mark a group of instructions for linker relaxation. The sequence of instructions within the group must be known to the linker so that relaxation can be performed.

'`.mode [mshort|mlong|mshort-double|mlong-double]`'

This directive specifies the ABI. It overrides the '`-mshort`', '`-mlong`', '`-mshort-double`' and '`-mlong-double`' options.

'`.far SYMBOL`'

This directive marks the symbol as a 'far' symbol meaning that it uses a '`call/rtc`' calling convention as opposed to '`jsr/rts`'. During a final link, the linker will identify references to the 'far' symbol and will verify the proper calling convention.

'`.interrupt SYMBOL`'

This directive marks the symbol as an interrupt entry point. This information is then used by the debugger to correctly unwind the frame across interrupts.

'`.xrefb SYMBOL`'

This directive is defined for compatibility with the 'Specification for Motorola 8 and 16-Bit Assembly Language Input Standard' and is ignored.

US

File: as.info, Node: M68HC11-Float, Next: M68HC11-opcodes, Prev: M68HC11-Directives, Up: M68HC11-Dependent

9.24.5 Floating Point

Packed decimal (P) format floating literals are not supported. Feel free to add the code!

The floating point formats generated by directives are these.

'`.float`'

'Single' precision floating point constants.

'`.double`'
 'Double' precision floating point constants.

'`.extend`'
 '`.ldouble`'
 'Extended' precision ('long double') floating point constants.



File: as.info, Node: M68HC11-opcodes, Prev: M68HC11-Float, Up: M68HC11-Dependent

9.24.6 Opcodes

* Menu:

* M68HC11-Branch:: Branch Improvement



File: as.info, Node: M68HC11-Branch, Up: M68HC11-opcodes

9.24.6.1 Branch Improvement

.....

Certain pseudo opcodes are permitted for branch instructions. They expand to the shortest branch instruction that reach the target. Generally these mnemonics are made by prepending 'j' to the start of Motorola mnemonic. These pseudo opcodes are not affected by the '`--short-branches`' or '`--force-long-branches`' options.

The following table summarizes the pseudo-operations.

Op	Displacement Width			
	Options			
	<code>--short-branches</code>		<code>--force-long-branches</code>	
	BYTE	WORD	BYTE	WORD
<code>bsr</code>	<code>bsr <pc-rel></code>	<code><error></code>		<code>jsr <abs></code>
<code>bra</code>	<code>bra <pc-rel></code>	<code><error></code>		<code>jmp <abs></code>
<code>jbsr</code>	<code>bsr <pc-rel></code>	<code>jsr <abs></code>	<code>bsr <pc-rel></code>	<code>jsr <abs></code>
<code>jbra</code>	<code>bra <pc-rel></code>	<code>jmp <abs></code>	<code>bra <pc-rel></code>	<code>jmp <abs></code>
<code>bXX</code>	<code>bXX <pc-rel></code>	<code><error></code>		<code>bNX +3; jmp <abs></code>
<code>jbXX</code>	<code>bXX <pc-rel></code>	<code>bNX +3; jmp <abs></code>	<code>bXX <pc-rel></code>	<code>bNX +3; jmp <abs></code>

XX: condition

NX: negative of condition XX

'`jbsr`'
 '`jbra`'

These are the simplest jump pseudo-operations; they always map to one particular machine instruction, depending on the displacement to the branch target.

'`jbXX`'
 Here, '`jbXX`' stands for an entire family of pseudo-operations,

where XX is a conditional branch or condition-code test. The full list of pseudo-ops in this family is:

```

    jbcc  jbeq  jbge  jbgt  jbhi  jbvs  jbpl  jblo
    jbcs  jbne  jblt  jble  jbls  jbvc  jbmi

```

For the cases of non-PC relative displacements and long displacements, 'as' issues a longer code fragment in terms of NX, the opposite condition to XX. For example, for the non-PC relative case:

```

    jbXX foo
gives
    bNXs oof
    jmp foo
oof:

```



File: as.info, Node: Meta-Dependent, Next: MicroBlaze-Dependent, Prev: M68HC11-Dependent, Up: Machine Dependencies

9.25 Meta Dependent Features

=====

* Menu:

```

* Meta Options::          Options
* Meta Syntax::          Meta Assembler Syntax

```



File: as.info, Node: Meta Options, Next: Meta Syntax, Up: Meta-Dependent

9.25.1 Options

The Imagination Technologies Meta architecture is implemented in a number of versions, with each new version adding new features such as instructions and registers. For precise details of what instructions each core supports, please see the chip's technical reference manual.

The following table lists all available Meta options.

```

'-mcpu=metac11'
  Generate code for Meta 1.1.

'-mcpu=metac12'
  Generate code for Meta 1.2.

'-mcpu=metac21'
  Generate code for Meta 2.1.

'-mfpu=metac21'
  Allow code to use FPU hardware of Meta 2.1.

```



File: as.info, Node: Meta Syntax, Prev: Meta Options, Up: Meta-Dependent

9.25.2 Syntax

* Menu:

* Meta-Chars:: Special Characters
 * Meta-Regs:: Register Names



File: as.info, Node: Meta-Chars, Next: Meta-Regs, Up: Meta Syntax

9.25.2.1 Special Characters

.....

'!' is the line comment character.

You can use ';' instead of a newline to separate statements.

Since '\$' has no special meaning, you may use it in symbol names.



File: as.info, Node: Meta-Regs, Prev: Meta-Chars, Up: Meta Syntax

9.25.2.2 Register Names

.....

Registers can be specified either using their mnemonic names, such as 'D0Re0', or using the unit plus register number separated by a '.', such as 'D0.0'.



File: as.info, Node: MicroBlaze-Dependent, Next: MIPS-Dependent, Prev: Meta-Dependent, Up: Machine Dependencies

9.26 MicroBlaze Dependent Features

=====

The Xilinx MicroBlaze processor family includes several variants, all using the same core instruction set. This chapter covers features of the GNU assembler that are specific to the MicroBlaze architecture. For details about the MicroBlaze instruction set, please see the 'MicroBlaze Processor Reference Guide (UG081)' available at www.xilinx.com.

* Menu:

* MicroBlaze Directives:: Directives for MicroBlaze Processors.
 * MicroBlaze Syntax:: Syntax for the MicroBlaze



File: as.info, Node: MicroBlaze Directives, Next: MicroBlaze Syntax, Up: MicroBlaze-Dependent

9.26.1 Directives

A number of assembler directives are available for MicroBlaze.

'.data8 EXPRESSION,...'

This directive is an alias for '.byte'. Each expression is assembled into an eight-bit value.

'.data16 EXPRESSION,...'

This directive is an alias for '.hword'. Each expression is

assembled into an 16-bit value.

`'.data32 EXPRESSION,...'`

This directive is an alias for `'.word'`. Each expression is assembled into an 32-bit value.

`'.ent NAME[,LABEL]'`

This directive is an alias for `'.func'` denoting the start of function NAME at (optional) LABEL.

`'.end NAME[,LABEL]'`

This directive is an alias for `'.endfunc'` denoting the end of function NAME.

`'.gpword LABEL,...'`

This directive is an alias for `'.rva'`. The resolved address of LABEL is stored in the data section.

`'.weakext LABEL'`

Declare that LABEL is a weak external symbol.

`'.rodata'`

Switch to `.rodata` section. Equivalent to `'.section .rodata'`

`'.sdata2'`

Switch to `.sdata2` section. Equivalent to `'.section .sdata2'`

`'.sdata'`

Switch to `.sdata` section. Equivalent to `'.section .sdata'`

`'.bss'`

Switch to `.bss` section. Equivalent to `'.section .bss'`

`'.sbss'`

Switch to `.sbss` section. Equivalent to `'.section .sbss'`



File: as.info, Node: MicroBlaze Syntax, Prev: MicroBlaze Directives, Up: MicroBlaze-Dependent

9.26.2 Syntax for the MicroBlaze

* Menu:

* MicroBlaze-Chars:: Special Characters



File: as.info, Node: MicroBlaze-Chars, Up: MicroBlaze Syntax

9.26.2.1 Special Characters

.....

The presence of a '#' on a line indicates the start of a comment that extends to the end of the current line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (`*note Comments::`) or a preprocessor control

command (*note Preprocessing:).

The ';' character can be used to separate statements on the same line.

US

File: as.info, Node: MIPS-Dependent, Next: MMIX-Dependent, Prev: MicroBlaze-Dependent, Up: Machine Dependencies

9.27 MIPS Dependent Features

=====

GNU 'as' for MIPS architectures supports several different MIPS processors, and MIPS ISA levels I through V, MIPS32, and MIPS64. For information about the MIPS instruction set, see 'MIPS RISC Architecture', by Kane and Heindrich (Prentice-Hall). For an overview of MIPS assembly conventions, see "Appendix D: Assembly Language Programming" in the same work.

* Menu:

- * MIPS Options:: Assembler options
- * MIPS Macros:: High-level assembly macros
- * MIPS Symbol Sizes:: Directives to override the size of symbols
- * MIPS Small Data:: Controlling the use of small data accesses
- * MIPS ISA:: Directives to override the ISA level
- * MIPS assembly options:: Directives to control code generation
- * MIPS autoextend:: Directives for extending MIPS 16 bit instructions
- * MIPS insn:: Directive to mark data as an instruction
- * MIPS FP ABIs:: Marking which FP ABI is in use
- * MIPS NaN Encodings:: Directives to record which NaN encoding is being used
- * MIPS Option Stack:: Directives to save and restore options
- * MIPS ASE Instruction Generation Overrides:: Directives to control generation of MIPS ASE instructions
- * MIPS Floating-Point:: Directives to override floating-point options
- * MIPS Syntax:: MIPS specific syntactical considerations

US

File: as.info, Node: MIPS Options, Next: MIPS Macros, Up: MIPS-Dependent

9.27.1 Assembler options

The MIPS configurations of GNU 'as' support these special options:

'-G NUM'

Set the "small data" limit to N bytes. The default limit is 8 bytes. *Note Controlling the use of small data accesses: MIPS Small Data.

'-EB'

'-EL'

Any MIPS configuration of 'as' can select big-endian or little-endian output at run time (unlike the other GNU development tools, which must be configured for one or the other). Use '-EB' to select big-endian output, and '-EL' for little-endian.

'-KPIC'

Generate SVR4-style PIC. This option tells the assembler to

generate SVR4-style position-independent macro expansions. It also tells the assembler to mark the output file as PIC.

'-mvxworks-pic'

Generate VxWorks PIC. This option tells the assembler to generate VxWorks-style position-independent macro expansions.

'-mips1'

'-mips2'

'-mips3'

'-mips4'

'-mips5'

'-mips32'

'-mips32r2'

'-mips32r3'

'-mips32r5'

'-mips32r6'

'-mips64'

'-mips64r2'

'-mips64r3'

'-mips64r5'

'-mips64r6'

Generate code for a particular MIPS Instruction Set Architecture level. '-mips1' corresponds to the R2000 and R3000 processors, '-mips2' to the R6000 processor, '-mips3' to the R4000 processor, and '-mips4' to the R8000 and R10000 processors. '-mips5', '-mips32', '-mips32r2', '-mips32r3', '-mips32r5', '-mips32r6', '-mips64', '-mips64r2', '-mips64r3', '-mips64r5', and '-mips64r6' correspond to generic MIPS V, MIPS32, MIPS32 Release 2, MIPS32 Release 3, MIPS32 Release 5, MIPS32 Release 6, MIPS64, and MIPS64 Release 2, MIPS64 Release 3, MIPS64 Release 5, and MIPS64 Release 6 ISA processors, respectively. You can also switch instruction sets during the assembly; see *note Directives to override the ISA level: MIPS ISA.

'-mgp32'

'-mfp32'

Some macros have different expansions for 32-bit and 64-bit registers. The register sizes are normally inferred from the ISA and ABI, but these flags force a certain group of registers to be treated as 32 bits wide at all times. '-mgp32' controls the size of general-purpose registers and '-mfp32' controls the size of floating-point registers.

The '.set gp=32' and '.set fp=32' directives allow the size of registers to be changed for parts of an object. The default value is restored by '.set gp=default' and '.set fp=default'.

On some MIPS variants there is a 32-bit mode flag; when this flag is set, 64-bit instructions generate a trap. Also, some 32-bit OSes only save the 32-bit registers on a context switch, so it is essential never to use the 64-bit registers.

'-mgp64'

'-mfp64'

Assume that 64-bit registers are available. This is provided in the interests of symmetry with '-mgp32' and '-mfp32'.

The '.set gp=64' and '.set fp=64' directives allow the size of

registers to be changed for parts of an object. The default value is restored by `'.set gp=default'` and `'.set fp=default'`.

`'-mfpxx'`

Make no assumptions about whether 32-bit or 64-bit floating-point registers are available. This is provided to support having modules compatible with either `'-mfp32'` or `'-mfp64'`. This option can only be used with MIPS II and above.

The `'.set fp=xx'` directive allows a part of an object to be marked as not making assumptions about 32-bit or 64-bit FP registers. The default value is restored by `'.set fp=default'`.

`'-modd-spreg'`

`'-mno-odd-spreg'`

Enable use of floating-point operations on odd-numbered single-precision registers when supported by the ISA. `'-mfpxx'` implies `'-mno-odd-spreg'`, otherwise the default is `'-modd-spreg'`

`'-mips16'`

`'-no-mips16'`

Generate code for the MIPS 16 processor. This is equivalent to putting `'.set mips16'` at the start of the assembly file.

`'-no-mips16'` turns off this option.

`'-mmicromips'`

`'-mno-micromips'`

Generate code for the microMIPS processor. This is equivalent to putting `'.set micromips'` at the start of the assembly file.

`'-mno-micromips'` turns off this option. This is equivalent to putting `'.set nomicromips'` at the start of the assembly file.

`'-msmartmips'`

`'-mno-smartmips'`

Enables the SmartMIPS extensions to the MIPS32 instruction set, which provides a number of new instructions which target smartcard and cryptographic applications. This is equivalent to putting `'.set smartmips'` at the start of the assembly file.

`'-mno-smartmips'` turns off this option.

`'-mips3d'`

`'-no-mips3d'`

Generate code for the MIPS-3D Application Specific Extension. This tells the assembler to accept MIPS-3D instructions. `'-no-mips3d'` turns off this option.

`'-mdmx'`

`'-no-mdmx'`

Generate code for the MDMX Application Specific Extension. This tells the assembler to accept MDMX instructions. `'-no-mdmx'` turns off this option.

`'-mdsp'`

`'-mno-dsp'`

Generate code for the DSP Release 1 Application Specific Extension. This tells the assembler to accept DSP Release 1 instructions.

`'-mno-dsp'` turns off this option.

`'-mdspr2'`

'-mno-dspr2'

Generate code for the DSP Release 2 Application Specific Extension. This option implies '-mdsp'. This tells the assembler to accept DSP Release 2 instructions. '-mno-dspr2' turns off this option.

'-mdspr3'

'-mno-dspr3'

Generate code for the DSP Release 3 Application Specific Extension. This option implies '-mdsp' and '-mdspr2'. This tells the assembler to accept DSP Release 3 instructions. '-mno-dspr3' turns off this option.

'-mmt'

'-mno-mt'

Generate code for the MT Application Specific Extension. This tells the assembler to accept MT instructions. '-mno-mt' turns off this option.

'-mmcu'

'-mno-mcu'

Generate code for the MCU Application Specific Extension. This tells the assembler to accept MCU instructions. '-mno-mcu' turns off this option.

'-mmsa'

'-mno-msa'

Generate code for the MIPS SIMD Architecture Extension. This tells the assembler to accept MSA instructions. '-mno-msa' turns off this option.

'-mxpa'

'-mno-xpa'

Generate code for the MIPS eXtended Physical Address (XPA) Extension. This tells the assembler to accept XPA instructions. '-mno-xpa' turns off this option.

'-mvirt'

'-mno-virt'

Generate code for the Virtualization Application Specific Extension. This tells the assembler to accept Virtualization instructions. '-mno-virt' turns off this option.

'-minsn32'

'-mno-insn32'

Only use 32-bit instruction encodings when generating code for the microMIPS processor. This option inhibits the use of any 16-bit instructions. This is equivalent to putting '.set insn32' at the start of the assembly file. '-mno-insn32' turns off this option. This is equivalent to putting '.set noinsn32' at the start of the assembly file. By default '-mno-insn32' is selected, allowing all instructions to be used.

'-mfix7000'

'-mno-fix7000'

Cause nops to be inserted if the read of the destination register of an mfhi or mflo instruction occurs in the following two instructions.

'-mfix-rm7000'

'-mno-fix-rm7000'

Cause nops to be inserted if a `dmult` or `dmultu` instruction is followed by a load instruction.

'-mfix-loongson2f-jump'

'-mno-fix-loongson2f-jump'

Eliminate instruction fetch from outside 256M region to work around the Loongson2F 'jump' instructions. Without it, under extreme cases, the kernel may crash. The issue has been solved in latest processor batches, but this fix has no side effect to them.

'-mfix-loongson2f-nop'

'-mno-fix-loongson2f-nop'

Replace nops by 'or at,at,zero' to work around the Loongson2F 'nop' errata. Without it, under extreme cases, the CPU might deadlock. The issue has been solved in later Loongson2F batches, but this fix has no side effect to them.

'-mfix-vr4120'

'-mno-fix-vr4120'

Insert nops to work around certain VR4120 errata. This option is intended to be used on GCC-generated code: it is not designed to catch all problems in hand-written assembler code.

'-mfix-vr4130'

'-mno-fix-vr4130'

Insert nops to work around the VR4130 'mflo'/'mfhi' errata.

'-mfix-24k'

'-mno-fix-24k'

Insert nops to work around the 24K 'eret'/'deret' errata.

'-mfix-cn63xxp1'

'-mno-fix-cn63xxp1'

Replace 'pref' hints 0 - 4 and 6 - 24 with hint 28 to work around certain CN63XXP1 errata.

'-m4010'

'-no-m4010'

Generate code for the LSI R4010 chip. This tells the assembler to accept the R4010-specific instructions ('addciu', 'ffc', etc.), and to not schedule 'nop' instructions around accesses to the 'HI' and 'LO' registers. '-no-m4010' turns off this option.

'-m4650'

'-no-m4650'

Generate code for the MIPS R4650 chip. This tells the assembler to accept the 'mad' and 'madu' instruction, and to not schedule 'nop' instructions around accesses to the 'HI' and 'LO' registers. '-no-m4650' turns off this option.

'-m3900'

'-no-m3900'

'-m4100'

'-no-m4100'

For each option '-mNNNN', generate code for the MIPS RNNNN chip. This tells the assembler to accept instructions specific to that chip, and to schedule for that chip's hazards.

'-march=CPU'

Generate code for a particular MIPS CPU. It is exactly equivalent to '-mCPU', except that there are more value of CPU understood. Valid CPU value are:

2000, 3000, 3900, 4000, 4010, 4100, 4111, vr4120, vr4130, vr4181, 4300, 4400, 4600, 4650, 5000, rm5200, rm5230, rm5231, rm5261, rm5721, vr5400, vr5500, 6000, rm7000, 8000, rm9000, 10000, 12000, 14000, 16000, 4kc, 4km, 4kp, 4ksc, 4kec, 4kem, 4kep, 4ksd, m4k, m4kp, m14k, m14kc, m14ke, m14kec, 24kc, 24kf2_1, 24kf, 24kf1_1, 24kec, 24kef2_1, 24kef, 24kef1_1, 34kc, 34kf2_1, 34kf, 34kf1_1, 34kn, 74kc, 74kf2_1, 74kf, 74kf1_1, 74kf3_2, 1004kc, 1004kf2_1, 1004kf, 1004kf1_1, interactiv, m5100, m5101, p5600, 5kc, 5kf, 20kc, 25kf, sb1, sb1a, i6400, p6600, loongson2e, loongson2f, loongson3a, octeon, octeon+, octeon2, octeon3, xlr, xlp

For compatibility reasons, 'Nx' and 'Bfx' are accepted as synonyms for 'Nf1_1'. These values are deprecated.

'-mtune=CPU'

Schedule and tune for a particular MIPS CPU. Valid CPU values are identical to '-march=CPU'.

'-mabi=ABI'

Record which ABI the source code uses. The recognized arguments are: '32', 'n32', 'o64', '64' and 'eabi'.

'-msym32'**'-mno-sym32'**

Equivalent to adding '.set sym32' or '.set nosym32' to the beginning of the assembler input. *Note MIPS Symbol Sizes::.

'-nocpp'

This option is ignored. It is accepted for command-line compatibility with other assemblers, which use it to turn off C style preprocessing. With GNU 'as', there is no need for '-nocpp', because the GNU assembler itself never runs the C preprocessor.

'-msoft-float'**'-mhard-float'**

Disable or enable floating-point instructions. Note that by default floating-point instructions are always allowed even with CPU targets that don't have support for these instructions.

'-msingle-float'**'-mdouble-float'**

Disable or enable double-precision floating-point operations. Note that by default double-precision floating-point operations are always allowed even with CPU targets that don't have support for these operations.

'--construct-floats'**'--no-construct-floats'**

The '--no-construct-floats' option disables the construction of double width floating point constants by loading the two halves of the value into the two single width floating point registers that make up the double width register. This feature is useful if the processor support the FR bit in its status register, and this bit

is known (by the programmer) to be set. This bit prevents the aliasing of the double width register by the single width registers.

By default '--construct-floats' is selected, allowing construction of these floating point constants.

'--relax-branch'

'--no-relax-branch'

The '--relax-branch' option enables the relaxation of out-of-range branches. Any branches whose target cannot be reached directly are converted to a small instruction sequence including an inverse-condition branch to the physically next instruction, and a jump to the original target is inserted between the two instructions. In PIC code the jump will involve further instructions for address calculation.

The 'BC1ANY2F', 'BC1ANY2T', 'BC1ANY4F', 'BC1ANY4T', 'BPOSGE32' and 'BPOSGE64' instructions are excluded from relaxation, because they have no complementing counterparts. They could be relaxed with the use of a longer sequence involving another branch, however this has not been implemented and if their target turns out of reach, they produce an error even if branch relaxation is enabled.

Also no MIPS16 branches are ever relaxed.

By default '--no-relax-branch' is selected, causing any out-of-range branches to produce an error.

'-mignore-branch-isa'

'-mno-ignore-branch-isa'

Ignore branch checks for invalid transitions between ISA modes.

The semantics of branches does not provide for an ISA mode switch, so in most cases the ISA mode a branch has been encoded for has to be the same as the ISA mode of the branch's target label. If the ISA modes do not match, then such a branch, if taken, will cause the ISA mode to remain unchanged and instructions that follow will be executed in the wrong ISA mode causing the program to misbehave or crash.

In the case of the 'BAL' instruction it may be possible to relax it to an equivalent 'JALX' instruction so that the ISA mode is switched at the run time as required. For other branches no relaxation is possible and therefore GAS has checks implemented that verify in branch assembly that the two ISA modes match, and report an error otherwise so that the problem with code can be diagnosed at the assembly time rather than at the run time.

However some assembly code, including generated code produced by some versions of GCC, may incorrectly include branches to data labels, which appear to require a mode switch but are either dead or immediately followed by valid instructions encoded for the same ISA the branch has been encoded for. While not strictly correct at the source level such code will execute as intended, so to help with these cases '-mignore-branch-isa' is supported which disables ISA mode checks for branches.

By default '-mno-ignore-branch-isa' is selected, causing any

invalid branch requiring a transition between ISA modes to produce an error.

'-mnan=ENCODING'

This option indicates whether the source code uses the IEEE 2008 NaN encoding ('-mnan=2008') or the original MIPS encoding ('-mnan=legacy'). It is equivalent to adding a '.nan' directive to the beginning of the source file. *Note MIPS NaN Encodings::.

'-mnan=legacy' is the default if no '-mnan' option or '.nan' directive is used.

'--trap'

'--no-break'

'as' automatically macro expands certain division and multiplication instructions to check for overflow and division by zero. This option causes 'as' to generate code to take a trap exception rather than a break exception when an error is detected. The trap instructions are only supported at Instruction Set Architecture level 2 and higher.

'--break'

'--no-trap'

Generate code to take a break exception rather than a trap exception when an error is detected. This is the default.

'-mpdr'

'-mno-pdr'

Control generation of '.pdr' sections. Off by default on IRIX, on elsewhere.

'-mshared'

'-mno-shared'

When generating code using the Unix calling conventions (selected by '-KPIC' or '-mcall_shared'), gas will normally generate code which can go into a shared library. The '-mno-shared' option tells gas to generate code which uses the calling convention, but can not go into a shared library. The resulting code is slightly more efficient. This option only affects the handling of the '.cpload' and '.cpsetup' pseudo-ops.



File: as.info, Node: MIPS Macros, Next: MIPS Symbol Sizes, Prev: MIPS Options, Up: MIPS-Dependent

9.27.2 High-level assembly macros

MIPS assemblers have traditionally provided a wider range of instructions than the MIPS architecture itself. These extra instructions are usually referred to as "macro" instructions (1).

Some MIPS macro instructions extend an underlying architectural instruction while others are entirely new. An example of the former type is 'and', which allows the third operand to be either a register or an arbitrary immediate value. Examples of the latter type include 'bgt', which branches to the third operand when the first operand is greater than the second operand, and 'ulh', which implements an unaligned 2-byte load.

One of the most common extensions provided by macros is to expand memory offsets to the full address range (32 or 64 bits) and to allow symbolic offsets such as 'my_data + 4' to be used in place of integer constants. For example, the architectural instruction 'lbu' allows only a signed 16-bit offset, whereas the macro 'lbu' allows code such as 'lbu \$4,array+32769(\$5)'. The implementation of these symbolic offsets depends on several factors, such as whether the assembler is generating SVR4-style PIC (selected by '-KPIC', *note Assembler options: MIPS Options.), the size of symbols (*note Directives to override the size of symbols: MIPS Symbol Sizes.), and the small data limit (*note Controlling the use of small data accesses: MIPS Small Data.).

Sometimes it is undesirable to have one assembly instruction expand to several machine instructions. The directive '.set nomacro' tells the assembler to warn when this happens. '.set macro' restores the default behavior.

Some macro instructions need a temporary register to store intermediate results. This register is usually '\$1', also known as '\$at', but it can be changed to any core register REG using '.set at=REG'. Note that '\$at' always refers to '\$1' regardless of which register is being used as the temporary register.

Implicit uses of the temporary register in macros could interfere with explicit uses in the assembly code. The assembler therefore warns whenever it sees an explicit use of the temporary register. The directive '.set noat' silences this warning while '.set at' restores the default behavior. It is safe to use '.set noat' while '.set nomacro' is in effect since single-instruction macros never need a temporary register.

Note that while the GNU assembler provides these macros for compatibility, it does not make any attempt to optimize them with the surrounding code.

----- Footnotes -----

(1) The term "macro" is somewhat overloaded here, since these macros have no relation to those defined by '.macro', *note '.macro': Macro.



File: as.info, Node: MIPS Symbol Sizes, Next: MIPS Small Data, Prev: MIPS Macros, Up: MIPS-Dependent

9.27.3 Directives to override the size of symbols

The n64 ABI allows symbols to have any 64-bit value. Although this provides a great deal of flexibility, it means that some macros have much longer expansions than their 32-bit counterparts. For example, the non-PIC expansion of 'dla \$4,sym' is usually:

```
lui    $4,%highest(sym)
lui    $1,%hi(sym)
daddiu $4,$4,%higher(sym)
daddiu $1,$1,%lo(sym)
dshll32 $4,$4,0
daddu  $4,$4,$1
```


whereas the 32-bit expansion is simply:

```
lui    $4,%hi(sym)
daddiu $4,$4,%lo(sym)
```

n64 code is sometimes constructed in such a way that all symbolic constants are known to have 32-bit values, and in such cases, it's preferable to use the 32-bit expansion instead of the 64-bit expansion.

You can use the `'.set sym32'` directive to tell the assembler that, from this point on, all expressions of the form `'SYMBOL'` or `'SYMBOL + OFFSET'` have 32-bit values. For example:

```
.set sym32
dla    $4,sym
lw     $4,sym+16
sw     $4,sym+0x8000($4)
```

will cause the assembler to treat `'sym'`, `'sym+16'` and `'sym+0x8000'` as 32-bit values. The handling of non-symbolic addresses is not affected.

The directive `'.set nosym32'` ends a `'.set sym32'` block and reverts to the normal behavior. It is also possible to change the symbol size using the command-line options `'-msym32'` and `'-mno-sym32'`.

These options and directives are always accepted, but at present, they have no effect for anything other than n64.



File: as.info, Node: MIPS Small Data, Next: MIPS ISA, Prev: MIPS Symbol Sizes, Up: MIPS-Dependent

9.27.4 Controlling the use of small data accesses

It often takes several instructions to load the address of a symbol. For example, when `'addr'` is a 32-bit symbol, the non-PIC expansion of `'dla $4,addr'` is usually:

```
lui    $4,%hi(addr)
daddiu $4,$4,%lo(addr)
```

The sequence is much longer when `'addr'` is a 64-bit symbol. *Note Directives to override the size of symbols: MIPS Symbol Sizes.

In order to cut down on this overhead, most embedded MIPS systems set aside a 64-kilobyte "small data" area and guarantee that all data of size `N` and smaller will be placed in that area. The limit `N` is passed to both the assembler and the linker using the command-line option `'-G N'`, *note Assembler options: MIPS Options. Note that the same value of `N` must be used when linking and when assembling all input files to the link; any inconsistency could cause a relocation overflow error.

The size of an object in the `'.bss'` section is set by the `'.comm'` or `'.lcomm'` directive that defines it. The size of an external object may be set with the `'.extern'` directive. For example, `'.extern sym,4'` declares that the object at `'sym'` is 4 bytes in length, while leaving `'sym'` otherwise undefined.

When no '-G' option is given, the default limit is 8 bytes. The option '-G 0' prevents any data from being automatically classified as small.

It is also possible to mark specific objects as small by putting them in the special sections '.sdata' and '.sbss', which are "small" counterparts of '.data' and '.bss' respectively. The toolchain will treat such data as small regardless of the '-G' setting.

On startup, systems that support a small data area are expected to initialize register '\$28', also known as '\$gp', in such a way that small data can be accessed using a 16-bit offset from that register. For example, when 'addr' is small data, the 'dla \$4,addr' instruction above is equivalent to:

```
daddiu $4,$28,%gp_rel(addr)
```

Small data is not supported for SVR4-style PIC.

US

File: as.info, Node: MIPS ISA, Next: MIPS assembly options, Prev: MIPS Small Data, Up: MIPS-Dependent

9.27.5 Directives to override the ISA level

GNU 'as' supports an additional directive to change the MIPS Instruction Set Architecture level on the fly: '.set mipsN'. N should be a number from 0 to 5, or 32, 32r2, 32r3, 32r5, 32r6, 64, 64r2, 64r3, 64r5 or 64r6. The values other than 0 make the assembler accept instructions for the corresponding ISA level, from that point on in the assembly. '.set mipsN' affects not only which instructions are permitted, but also how certain macros are expanded. '.set mips0' restores the ISA level to its original level: either the level you selected with command line options, or the default for your configuration. You can use this feature to permit specific MIPS III instructions while assembling in 32 bit mode. Use this directive with care!

The '.set arch=CPU' directive provides even finer control. It changes the effective CPU target and allows the assembler to use instructions specific to a particular CPU. All CPUs supported by the '-march' command line option are also selectable by this directive. The original value is restored by '.set arch=default'.

The directive '.set mips16' puts the assembler into MIPS 16 mode, in which it will assemble instructions for the MIPS 16 processor. Use '.set nomips16' to return to normal 32 bit mode.

Traditional MIPS assemblers do not support this directive.

The directive '.set micromips' puts the assembler into microMIPS mode, in which it will assemble instructions for the microMIPS processor. Use '.set nomicromips' to return to normal 32 bit mode.

Traditional MIPS assemblers do not support this directive.

US

File: as.info, Node: MIPS assembly options, Next: MIPS autoextend, Prev: MIPS ISA,

File: as.info, Node: MIPS assembly options, Next: MIPS autoextend, Prev: MIPS ISA, Up: MIPS-Dependent

9.27.6 Directives to control code generation

The `.module` directive allows command line options to be set directly from assembly. The format of the directive matches the `.set` directive but only those options which are relevant to a whole module are supported. The effect of a `.module` directive is the same as the corresponding command line option. Where `.set` directives support returning to a default then the `.module` directives do not as they define the defaults.

These module-level directives must appear first in assembly.

Traditional MIPS assemblers do not support this directive.

The directive `.set insn32` makes the assembler only use 32-bit instruction encodings when generating code for the microMIPS processor. This directive inhibits the use of any 16-bit instructions from that point on in the assembly. The `.set noinsn32` directive allows 16-bit instructions to be accepted.

Traditional MIPS assemblers do not support this directive.



File: as.info, Node: MIPS autoextend, Next: MIPS insn, Prev: MIPS assembly options, Up: MIPS-Dependent

9.27.7 Directives for extending MIPS 16 bit instructions

By default, MIPS 16 instructions are automatically extended to 32 bits when necessary. The directive `.set noautoextend` will turn this off. When `.set noautoextend` is in effect, any 32 bit instruction must be explicitly extended with the `.e` modifier (e.g., `li.e $4,1000`). The directive `.set autoextend` may be used to once again automatically extend instructions when necessary.

This directive is only meaningful when in MIPS 16 mode. Traditional MIPS assemblers do not support this directive.



File: as.info, Node: MIPS insn, Next: MIPS FP ABIs, Prev: MIPS autoextend, Up: MIPS-Dependent

9.27.8 Directive to mark data as an instruction

The `.insn` directive tells `as` that the following data is actually instructions. This makes a difference in MIPS 16 and microMIPS modes: when loading the address of a label which precedes instructions, `as` automatically adds 1 to the value, so that jumping to the loaded address will do the right thing.

The `.global` and `.globl` directives supported by `as` will by default mark the symbol as pointing to a region of data not code. This means that, for example, any instructions following such a symbol will

not be disassembled by 'objdump' as it will regard them as data. To change this behavior an optional section name can be placed after the symbol name in the '.global' directive. If this section exists and is known to be a code section, then the symbol will be marked as pointing at code not data. Ie the syntax for the directive is:

```
'.global SYMBOL[ SECTION][, SYMBOL[ SECTION]] ...',
```

Here is a short example:

```

        .global foo .text, bar, baz .data
foo:
        nop
bar:
        .word 0x0
baz:
        .word 0x1

```



File: as.info, Node: MIPS FP ABIs, Next: MIPS NaN Encodings, Prev: MIPS insn, Up: MIPS-Dependent

9.27.9 Directives to control the FP ABI

* Menu:

* MIPS FP ABI History::	History of FP ABIs
* MIPS FP ABI Variants::	Supported FP ABIs
* MIPS FP ABI Selection::	Automatic selection of FP ABI
* MIPS FP ABI Compatibility::	Linking different FP ABI variants



File: as.info, Node: MIPS FP ABI History, Next: MIPS FP ABI Variants, Up: MIPS FP ABIs

9.27.9.1 History of FP ABIs

.....

The MIPS ABIs support a variety of different floating-point extensions where calling-convention and register sizes vary for floating-point data. The extensions exist to support a wide variety of optional architecture features. The resulting ABI variants are generally incompatible with each other and must be tracked carefully.

Traditionally the use of an explicit '.gnu_attribute 4, N' directive is used to indicate which ABI is in use by a specific module. It was then left to the user to ensure that command line options and the selected ABI were compatible with some potential for inconsistencies.



File: as.info, Node: MIPS FP ABI Variants, Next: MIPS FP ABI Selection, Prev: MIPS FP ABI History, Up: MIPS FP ABIs

9.27.9.2 Supported FP ABIs

.....

The supported floating-point ABI variants are:

- '0 - No floating-point'
This variant is used to indicate that floating-point is not used within the module at all and therefore has no impact on the ABI. This is the default.
- '1 - Double-precision'
This variant indicates that double-precision support is used. For 64-bit ABIs this means that 64-bit wide floating-point registers are required. For 32-bit ABIs this means that 32-bit wide floating-point registers are required and double-precision operations use pairs of registers.
- '2 - Single-precision'
This variant indicates that single-precision support is used. Double precision operations will be supported via soft-float routines.
- '3 - Soft-float'
This variant indicates that although floating-point support is used all operations are emulated in software. This means the ABI is modified to pass all floating-point data in general-purpose registers.
- '4 - Deprecated'
This variant existed as an initial attempt at supporting 64-bit wide floating-point registers for O32 ABI on a MIPS32r2 CPU. This has been superseded by 5, 6 and 7.
- '5 - Double-precision 32-bit CPU, 32-bit or 64-bit FPU'
This variant is used by 32-bit ABIs to indicate that the floating-point code in the module has been designed to operate correctly with either 32-bit wide or 64-bit wide floating-point registers. Double-precision support is used. Only O32 currently supports this variant and requires a minimum architecture of MIPS II.
- '6 - Double-precision 32-bit FPU, 64-bit FPU'
This variant is used by 32-bit ABIs to indicate that the floating-point code in the module requires 64-bit wide floating-point registers. Double-precision support is used. Only O32 currently supports this variant and requires a minimum architecture of MIPS32r2.
- '7 - Double-precision compat 32-bit FPU, 64-bit FPU'
This variant is used by 32-bit ABIs to indicate that the floating-point code in the module requires 64-bit wide floating-point registers. Double-precision support is used. This differs from the previous ABI as it restricts use of odd-numbered single-precision registers. Only O32 currently supports this variant and requires a minimum architecture of MIPS32r2.



File: as.info, Node: MIPS FP ABI Selection, Next: MIPS FP ABI Compatibility, Prev: MIPS FP ABI Variants, Up: MIPS FP ABIs

9.27.9.3 Automatic selection of FP ABI

.....

In order to simplify and add safety to the process of selecting the correct floating-point ABI, the assembler will automatically infer the correct `.gnu_attribute 4, N` directive based on command line options and `.module` overrides. Where an explicit `.gnu_attribute 4, N` directive has been seen then a warning will be raised if it does not match an inferred setting.

The floating-point ABI is inferred as follows. If `-msoft-float` has been used the module will be marked as soft-float. If `-msingle-float` has been used then the module will be marked as single-precision. The remaining ABIs are then selected based on the FP register width. Double-precision is selected if the width of GP and FP registers match and the special double-precision variants for 32-bit ABIs are then selected depending on `-mfpxx`, `-mfp64` and `-mno-odd-spreg`.



File: as.info, Node: MIPS FP ABI Compatibility, Prev: MIPS FP ABI Selection, Up: MIPS FP ABIs

9.27.9.4 Linking different FP ABI variants

.....

Modules using the default FP ABI (no floating-point) can be linked with any other (singular) FP ABI variant.

Special compatibility support exists for O32 with the four double-precision FP ABI variants. The `-mfpxx` FP ABI is specifically designed to be compatible with the standard double-precision ABI and the `-mfp64` FP ABIs. This makes it desirable for O32 modules to be built as `-mfpxx` to ensure the maximum compatibility with other modules produced for more specific needs. The only FP ABIs which cannot be linked together are the standard double-precision ABI and the full `-mfp64` ABI with `-modd-spreg`.



File: as.info, Node: MIPS NaN Encodings, Next: MIPS Option Stack, Prev: MIPS FP ABIs, Up: MIPS-Dependent

9.27.10 Directives to record which NaN encoding is being used

The IEEE 754 floating-point standard defines two types of not-a-number (NaN) data: "signalling" NaNs and "quiet" NaNs. The original version of the standard did not specify how these two types should be distinguished. Most implementations followed the i387 model, in which the first bit of the significand is set for quiet NaNs and clear for signalling NaNs. However, the original MIPS implementation assigned the opposite meaning to the bit, so that it was set for signalling NaNs and clear for quiet NaNs.

The 2008 revision of the standard formally suggested the i387 choice and as from Sep 2012 the current release of the MIPS architecture therefore optionally supports that form. Code that uses one NaN encoding would usually be incompatible with code that uses the other NaN encoding, so MIPS ELF objects have a flag (`'EF_MIPS_NAN2008'`) to record which encoding is being used.

Assembly files can use the `.nan` directive to select between the two encodings. `.nan 2008` says that the assembly file uses the IEEE

754-2008 encoding while `'.nan legacy'` says that the file uses the original MIPS encoding. If several `'.nan'` directives are given, the final setting is the one that is used.

The command-line options `'-mnan=legacy'` and `'-mnan=2008'` can be used instead of `'.nan legacy'` and `'.nan 2008'` respectively. However, any `'.nan'` directive overrides the command-line setting.

`'.nan legacy'` is the default if no `'.nan'` directive or `'-mnan'` option is given.

Note that GNU `'as'` does not produce NaNs itself and therefore these directives do not affect code generation. They simply control the setting of the `'EF_MIPS_NAN2008'` flag.

Traditional MIPS assemblers do not support these directives.



File: as.info, Node: MIPS Option Stack, Next: MIPS ASE Instruction Generation Overrides, Prev: MIPS NaN Encodings, Up: MIPS-Dependent

9.27.11 Directives to save and restore options

The directives `'.set push'` and `'.set pop'` may be used to save and restore the current settings for all the options which are controlled by `'.set'`. The `'.set push'` directive saves the current settings on a stack. The `'.set pop'` directive pops the stack and restores the settings.

These directives can be useful inside an macro which must change an option such as the ISA level or instruction reordering but does not want to change the state of the code which invoked the macro.

Traditional MIPS assemblers do not support these directives.



File: as.info, Node: MIPS ASE Instruction Generation Overrides, Next: MIPS Floating-Point, Prev: MIPS Option Stack, Up: MIPS-Dependent

9.27.12 Directives to control generation of MIPS ASE instructions

The directive `'.set mips3d'` makes the assembler accept instructions from the MIPS-3D Application Specific Extension from that point on in the assembly. The `'.set nomips3d'` directive prevents MIPS-3D instructions from being accepted.

The directive `'.set smartmips'` makes the assembler accept instructions from the SmartMIPS Application Specific Extension to the MIPS32 ISA from that point on in the assembly. The `'.set nosmartmips'` directive prevents SmartMIPS instructions from being accepted.

The directive `'.set mdmx'` makes the assembler accept instructions from the MDMX Application Specific Extension from that point on in the assembly. The `'.set nomdmx'` directive prevents MDMX instructions from being accepted.

The directive `'.set dsp'` makes the assembler accept instructions from

the DSP Release 1 Application Specific Extension from that point on in the assembly. The `.set nodsp` directive prevents DSP Release 1 instructions from being accepted.

The directive `.set dspr2` makes the assembler accept instructions from the DSP Release 2 Application Specific Extension from that point on in the assembly. This directive implies `.set dsp`. The `.set nodspr2` directive prevents DSP Release 2 instructions from being accepted.

The directive `.set dspr3` makes the assembler accept instructions from the DSP Release 3 Application Specific Extension from that point on in the assembly. This directive implies `.set dsp` and `.set dspr2`. The `.set nodspr3` directive prevents DSP Release 3 instructions from being accepted.

The directive `.set mt` makes the assembler accept instructions from the MT Application Specific Extension from that point on in the assembly. The `.set nomt` directive prevents MT instructions from being accepted.

The directive `.set mcu` makes the assembler accept instructions from the MCU Application Specific Extension from that point on in the assembly. The `.set nomcu` directive prevents MCU instructions from being accepted.

The directive `.set msa` makes the assembler accept instructions from the MIPS SIMD Architecture Extension from that point on in the assembly. The `.set nomsa` directive prevents MSA instructions from being accepted.

The directive `.set virt` makes the assembler accept instructions from the Virtualization Application Specific Extension from that point on in the assembly. The `.set novirt` directive prevents Virtualization instructions from being accepted.

The directive `.set xpa` makes the assembler accept instructions from the XPA Extension from that point on in the assembly. The `.set noxpa` directive prevents XPA instructions from being accepted.

Traditional MIPS assemblers do not support these directives.



File: as.info, Node: MIPS Floating-Point, Next: MIPS Syntax, Prev: MIPS ASE Instruction Generation Overrides, Up: MIPS-Dependent

9.27.13 Directives to override floating-point options

The directives `.set softfloat` and `.set hardfloat` provide finer control of disabling and enabling float-point instructions. These directives always override the default (that hard-float instructions are accepted) or the command-line options (`'-msoft-float'` and `'-mhard-float'`).

The directives `.set singlefloat` and `.set doublefloat` provide finer control of disabling and enabling double-precision float-point operations. These directives always override the default (that double-precision operations are accepted) or the command-line options (`'-msingle-float'` and `'-mdouble-float'`).

Traditional MIPS assemblers do not support these directives.



File: as.info, Node: MIPS Syntax, Prev: MIPS Floating-Point, Up: MIPS-Dependent

9.27.14 Syntactical considerations for the MIPS assembler

* Menu:

* MIPS-Chars:: Special Characters



File: as.info, Node: MIPS-Chars, Up: MIPS Syntax

9.27.14.1 Special Characters

.....

The presence of a '#' on a line indicates the start of a comment that extends to the end of the current line.

If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

The ';' character can be used to separate statements on the same line.



File: as.info, Node: MMIX-Dependent, Next: MSP430-Dependent, Prev: MIPS-Dependent, Up: Machine Dependencies

9.28 MMIX Dependent Features

=====

* Menu:

* MMIX-Opts:: Command-line Options
 * MMIX-Expand:: Instruction expansion
 * MMIX-Syntax:: Syntax
 * MMIX-mmixal:: Differences to 'mmixal' syntax and semantics



File: as.info, Node: MMIX-Opts, Next: MMIX-Expand, Up: MMIX-Dependent

9.28.1 Command-line Options

The MMIX version of 'as' has some machine-dependent options.

When '--fixed-special-register-names' is specified, only the register names specified in *note MMIX-Regs: are recognized in the instructions 'PUT' and 'GET'.

You can use the '--globalize-symbols' to make all symbols global. This option is useful when splitting up a 'mmixal' program into several files.

The '--gnu-syntax' turns off most syntax compatibility with 'mmixal'. Its usability is currently doubtful.

The '--relax' option is not fully supported, but will eventually make the object file prepared for linker relaxation.

If you want to avoid inadvertently calling a predefined symbol and would rather get an error, for example when using 'as' with a compiler or other machine-generated code, specify '--no-predefined-syms'. This turns off built-in predefined definitions of all such symbols, including rounding-mode symbols, segment symbols, 'BIT' symbols, and 'TRAP' symbols used in 'mmix' "system calls". It also turns off predefined special-register names, except when used in 'PUT' and 'GET' instructions.

By default, some instructions are expanded to fit the size of the operand or an external symbol (*note MMIX-Expand:). By passing '--no-expand', no such expansion will be done, instead causing errors at link time if the operand does not fit.

The 'mmixal' documentation (*note mmixsite:.) specifies that global registers allocated with the 'GREG' directive (*note MMIX-greg:.) and initialized to the same non-zero value, will refer to the same global register. This isn't strictly enforceable in 'as' since the final addresses aren't known until link-time, but it will do an effort unless the '--no-merge-gregs' option is specified. (Register merging isn't yet implemented in 'ld'.)

'as' will warn every time it expands an instruction to fit an operand unless the option '-x' is specified. It is believed that this behaviour is more useful than just mimicking 'mmixal's' behaviour, in which instructions are only expanded if the '-x' option is specified, and assembly fails otherwise, when an instruction needs to be expanded. It needs to be kept in mind that 'mmixal' is both an assembler and linker, while 'as' will expand instructions that at link stage can be contracted. (Though linker relaxation isn't yet implemented in 'ld'.) The option '-x' also implem '--linker-allocated-gregs'.

If instruction expansion is enabled, 'as' can expand a 'PUSHJ' instruction into a series of instructions. The shortest expansion is to not expand it, but just mark the call as redirectable to a stub, which 'ld' creates at link-time, but only if the original 'PUSHJ' instruction is found not to reach the target. The stub consists of the necessary instructions to form a jump to the target. This happens if 'as' can assert that the 'PUSHJ' instruction can reach such a stub. The option '--no-pushj-stubs' disables this shorter expansion, and the longer series of instructions is then created at assembly-time. The option '--no-stubs' is a synonym, intended for compatibility with future releases, where generation of stubs for other instructions may be implemented.

Usually a two-operand-expression (*note GREG-base:.) without a matching 'GREG' directive is treated as an error by 'as'. When the option '--linker-allocated-gregs' is in effect, they are instead passed through to the linker, which will allocate as many global registers as is needed.

File: as.info, Node: MMIX-Expand, Next: MMIX-Syntax, Prev: MMIX-Opts, Up: MMIX-Dependent

9.28.2 Instruction expansion

When 'as' encounters an instruction with an operand that is either not known or does not fit the operand size of the instruction, 'as' (and 'ld') will expand the instruction into a sequence of instructions semantically equivalent to the operand fitting the instruction. Expansion will take place for the following instructions:

- 'GETA'
- Expands to a sequence of four instructions: 'SETL', 'INCML', 'INCMH' and 'INCH'. The operand must be a multiple of four.
- Conditional branches
- A branch instruction is turned into a branch with the complemented condition and prediction bit over five instructions; four instructions setting '\$255' to the operand value, which like with 'GETA' must be a multiple of four, and a final 'GO \$255,\$255,0'.
- 'PUSHJ'
- Similar to expansion for conditional branches; four instructions set '\$255' to the operand value, followed by a 'PUSHGO \$255,\$255,0'.
- 'JMP'
- Similar to conditional branches and 'PUSHJ'. The final instruction is 'GO \$255,\$255,0'.

The linker 'ld' is expected to shrink these expansions for code assembled with '--relax' (though not currently implemented).



File: as.info, Node: MMIX-Syntax, Next: MMIX-mmixal, Prev: MMIX-Expand, Up: MMIX-Dependent

9.28.3 Syntax

The assembly syntax is supposed to be upward compatible with that described in Sections 1.3 and 1.4 of 'The Art of Computer Programming, Volume 1'. Draft versions of those chapters as well as other MMIX information is located at <http://www-cs-faculty.stanford.edu/~knuth/mmix-news.html>. Most code examples from the mmixal package located there should work unmodified when assembled and linked as single files, with a few noteworthy exceptions (*note MMIX-mmixal:).

Before an instruction is emitted, the current location is aligned to the next four-byte boundary. If a label is defined at the beginning of the line, its value will be the aligned value.

In addition to the traditional hex-prefix '0x', a hexadecimal number can also be specified by the prefix character '#'.

After all operands to an MMIX instruction or directive have been specified, the rest of the line is ignored, treated as a comment.

* Menu:

```
* MMIX-Chars::          Special Characters
* MMIX-Symbols::       Symbols
* MMIX-Regs::         Register Names
* MMIX-Pseudos::      Assembler Directives
```



File: as.info, Node: MMIX-Chars, Next: MMIX-Symbols, Up: MMIX-Syntax

9.28.3.1 Special Characters

.....

The characters '*' and '#' are line comment characters; each start a comment at the beginning of a line, but only at the beginning of a line. A '#' prefixes a hexadecimal number if found elsewhere on a line. If a '#' appears at the start of a line the whole line is treated as a comment, but the line can also act as a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

Two other characters, '%' and '!', each start a comment anywhere on the line. Thus you can't use the 'modulus' and 'not' operators in expressions normally associated with these two characters.

A ';' is a line separator, treated as a new-line, so separate instructions can be specified on a single line.



File: as.info, Node: MMIX-Symbols, Next: MMIX-Regs, Prev: MMIX-Chars, Up: MMIX-Syntax

9.28.3.2 Symbols

.....

The character ':' is permitted in identifiers. There are two exceptions to it being treated as any other symbol character: if a symbol begins with ':', it means that the symbol is in the global namespace and that the current prefix should not be prepended to that symbol (*note MMIX-prefix::). The ':' is then not considered part of the symbol. For a symbol in the label position (first on a line), a ':' at the end of a symbol is silently stripped off. A label is permitted, but not required, to be followed by a ':', as with many other assembly formats.

The character '@' in an expression, is a synonym for '.', the current location.

In addition to the common forward and backward local symbol formats (*note Symbol Names::), they can be specified with upper-case 'B' and 'F', as in '8B' and '9F'. A local label defined for the current position is written with a 'H' appended to the number:

```
3H LDB $0,$1,2
```

This and traditional local-label formats cannot be mixed: a label must be defined and referred to using the same format.

There's a minor caveat: just as for the ordinary local symbols, the local symbols are translated into ordinary symbols using control characters to hide the ordinal number of the symbol. Unfortunately, these symbols are not translated back in error messages. Thus you may see confusing error messages when local symbols are used. Control characters '\003' (control-C) and '\004' (control-D) are used for the

MMIX-specific local-symbol syntax.

The symbol 'Main' is handled specially; it is always global.

By defining the symbols '`__MMIX.start..text`' and '`__MMIX.start..data`', the address of respectively the '.text' and '.data' segments of the final program can be defined, though when linking more than one object file, the code or data in the object file containing the symbol is not guaranteed to be start at that position; just the final executable. *Note MMIX-loc:.



File: as.info, Node: MMIX-Regs, Next: MMIX-Pseudos, Prev: MMIX-Symbols, Up: MMIX-Syntax

9.28.3.3 Register names

.....

Local and global registers are specified as '\$0' to '\$255'. The recognized special register names are 'rJ', 'rA', 'rB', 'rC', 'rD', 'rE', 'rF', 'rG', 'rH', 'rI', 'rK', 'rL', 'rM', 'rN', 'rO', 'rP', 'rQ', 'rR', 'rS', 'rT', 'rU', 'rV', 'rW', 'rX', 'rY', 'rZ', 'rBB', 'rTT', 'rWW', 'rXX', 'rYY' and 'rZZ'. A leading ':' is optional for special register names.

Local and global symbols can be equated to register names and used in place of ordinary registers.

Similarly for special registers, local and global symbols can be used. Also, symbols equated from numbers and constant expressions are allowed in place of a special register, except when either of the options '--no-predefined-syms' and '--fixed-special-register-names' are specified. Then only the special register names above are allowed for the instructions having a special register operand; 'GET' and 'PUT'.



File: as.info, Node: MMIX-Pseudos, Prev: MMIX-Regs, Up: MMIX-Syntax

9.28.3.4 Assembler Directives

.....

'LOC'

The 'LOC' directive sets the current location to the value of the operand field, which may include changing sections. If the operand is a constant, the section is set to either '.data' if the value is '0x2000000000000000' or larger, else it is set to '.text'. Within a section, the current location may only be changed to monotonically higher addresses. A LOC expression must be a previously defined symbol or a "pure" constant.

An example, which sets the label PREV to the current location, and updates the current location to eight bytes forward:

```
prev LOC @+8
```

When a LOC has a constant as its operand, a symbol '`__MMIX.start..text`' or '`__MMIX.start..data`' is defined depending on the address as mentioned above. Each such symbol is interpreted as special by the linker, locating the section at that address.

Note that if multiple files are linked, the first object file with that section will be mapped to that address (not necessarily the file with the LOC definition).

'LOCAL'

Example:

```
LOCAL external_symbol
LOCAL 42
.local asymbol
```

This directive-operation generates a link-time assertion that the operand does not correspond to a global register. The operand is an expression that at link-time resolves to a register symbol or a number. A number is treated as the register having that number. There is one restriction on the use of this directive: the pseudo-directive must be placed in a section with contents, code or data.

'IS'

The 'IS' directive:

```
asymbol IS an_expression
```

sets the symbol 'asymbol' to 'an_expression'. A symbol may not be set more than once using this directive. Local labels may be set using this directive, for example:

```
5H IS @+4
```

'GREG'

This directive reserves a global register, gives it an initial value and optionally gives it a symbolic name. Some examples:

```
areg GREG
breg GREG data_value
  GREG data_buffer
  .greg creg, another_data_value
```

The symbolic register name can be used in place of a (non-special) register. If a value isn't provided, it defaults to zero. Unless the option '--no-merge-gregs' is specified, non-zero registers allocated with this directive may be eliminated by 'as'; another register with the same value used in its place. Any of the instructions 'CSWAP', 'GO', 'LDA', 'LDBU', 'LDB', 'LDHT', 'LDOU', 'LDO', 'LDSF', 'LDTU', 'LDT', 'LDUNC', 'LDVTS', 'LDWU', 'LDW', 'PREGO', 'PRELD', 'PREST', 'PUSHGO', 'STBU', 'STB', 'STCO', 'STHT', 'STOU', 'STSF', 'STTU', 'STT', 'STUNC', 'SYNCD', 'SYNCID', can have a value nearby an initial value in place of its second and third operands. Here, "nearby" is defined as within the range 0...255 from the initial value of such an allocated register.

```
buffer1 BYTE 0,0,0,0,0
buffer2 BYTE 0,0,0,0,0
...
GREG buffer1
LDOU $42,buffer2
```

In the example above, the 'Y' field of the 'LDOUI' instruction (LDOU with a constant Z) will be replaced with the global register allocated for 'buffer1', and the 'Z' field will have the value 5,

the offset from 'buffer1' to 'buffer2'. The result is equivalent to this code:

```
buffer1 BYTE 0,0,0,0,0
buffer2 BYTE 0,0,0,0,0
...
tmpreg GREG buffer1
LDOU $42,tmpreg,(buffer2-buffer1)
```

Global registers allocated with this directive are allocated in order higher-to-lower within a file. Other than that, the exact order of register allocation and elimination is undefined. For example, the order is undefined when more than one file with such directives are linked together. With the options '-x' and '--linker-allocated-gregs', 'GREG' directives for two-operand cases like the one mentioned above can be omitted. Sufficient global registers will then be allocated by the linker.

'BYTE'

The 'BYTE' directive takes a series of operands separated by a comma. If an operand is a string (*note Strings:), each character of that string is emitted as a byte. Other operands must be constant expressions without forward references, in the range 0...255. If you need operands having expressions with forward references, use '.byte' (*note Byte:). An operand can be omitted, defaulting to a zero value.

'WYDE'

'TETRA'

'OCTA'

The directives 'WYDE', 'TETRA' and 'OCTA' emit constants of two, four and eight bytes size respectively. Before anything else happens for the directive, the current location is aligned to the respective constant-size boundary. If a label is defined at the beginning of the line, its value will be that after the alignment. A single operand can be omitted, defaulting to a zero value emitted for the directive. Operands can be expressed as strings (*note Strings:), in which case each character in the string is emitted as a separate constant of the size indicated by the directive.

'PREFIX'

The 'PREFIX' directive sets a symbol name prefix to be prepended to all symbols (except local symbols, *note MMIX-Symbols:), that are not prefixed with ':', until the next 'PREFIX' directive. Such prefixes accumulate. For example,

```
PREFIX a
PREFIX b
c IS 0
```

defines a symbol 'abc' with the value 0.

'BSPEC'

'ESPEC'

A pair of 'BSPEC' and 'ESPEC' directives delimit a section of special contents (without specified semantics). Example:

```
BSPEC 42
TETRA 1,2,3
```

ESPEC

The single operand to 'BSPEC' must be number in the range 0...255.
The 'BSPEC' number 80 is used by the GNU binutils implementation.



File: as.info, Node: MMIX-mmixal, Prev: MMIX-Syntax, Up: MMIX-Dependent

9.28.4 Differences to 'mmixal'

The binutils 'as' and 'ld' combination has a few differences in function compared to 'mmixal' (*note mmixsite:).

The replacement of a symbol with a GREG-allocated register (*note GREG-base:.) is not handled the exactly same way in 'as' as in 'mmixal'. This is apparent in the 'mmixal' example file 'inout.mms', where different registers with different offsets, eventually yielding the same address, are used in the first instruction. This type of difference should however not affect the function of any program unless it has specific assumptions about the allocated register number.

Line numbers (in the 'mmo' object format) are currently not supported.

Expression operator precedence is not that of mmixal: operator precedence is that of the C programming language. It's recommended to use parentheses to explicitly specify wanted operator precedence whenever more than one type of operators are used.

The serialize unary operator '&', the fractional division operator '//', the logical not operator '!' and the modulus operator '%' are not available.

Symbols are not global by default, unless the option '--globalize-symbols' is passed. Use the '.global' directive to globalize symbols (*note Global:).

Operand syntax is a bit stricter with 'as' than 'mmixal'. For example, you can't say 'addu 1,2,3', instead you must write 'addu \$1,\$2,3'.

You can't LOC to a lower address than those already visited (i.e., "backwards").

A LOC directive must come before any emitted code.

Predefined symbols are visible as file-local symbols after use. (In the ELF file, that is--the linked mmo file has no notion of a file-local symbol.)

Some mapping of constant expressions to sections in LOC expressions is attempted, but that functionality is easily confused and should be avoided unless compatibility with 'mmixal' is required. A LOC expression to '0x2000000000000000' or higher, maps to the '.data' section and lower addresses map to the '.text' section (*note MMIX-loc:).

The code and data areas are each contiguous. Sparse programs with far-away LOC directives will take up the same amount of space as a

contiguous program with zeros filled in the gaps between the LOC directives. If you need sparse programs, you might try and get the wanted effect with a linker script and splitting up the code parts into sections (*note Section:). Assembly code for this, to be compatible with 'mmixal', would look something like:

```
.if 0
LOC away_expression
.else
.section away,"ax"
.fi
```

'as' will not execute the LOC directive and 'mmixal' ignores the lines with '.'. This construct can be used generally to help compatibility.

Symbols can't be defined twice-not even to the same value.

Instruction mnemonics are recognized case-insensitive, though the 'IS' and 'GREG' pseudo-operations must be specified in upper-case characters.

There's no unicode support.

The following is a list of programs in 'mmix.tar.gz', available at <<http://www-cs-faculty.stanford.edu/~knuth/mmix-news.html>>, last checked with the version dated 2001-08-25 (md5sum c393470cfc86fac040487d22d2bf0172) that assemble with 'mmixal' but do not assemble with 'as':

```
'silly.mms'
  LOC to a previous address.
'sim.mms'
  Redefines symbol 'Done'.
'test.mms'
  Uses the serial operator '&'.
```



File: as.info, Node: MSP430-Dependent, Next: NDS32-Dependent, Prev: MMIX-Dependent, Up: Machine Dependencies

9.29 MSP 430 Dependent Features

=====

* Menu:

```
* MSP430 Options::           Options
* MSP430 Syntax::           Syntax
* MSP430 Floating Point::   Floating Point
* MSP430 Directives::       MSP 430 Machine Directives
* MSP430 Opcodes::          Opcodes
* MSP430 Profiling Capability:: Profiling Capability
```



File: as.info, Node: MSP430 Options, Next: MSP430 Syntax, Up: MSP430-Dependent

9.29.1 Options

```
'-mmcu'
  selects the mcu architecture. If the architecture is 430Xv2 then
```

this also enables NOP generation unless the '-mN' is also specified.

'-mcpu'

selects the cpu architecture. If the architecture is 430Xv2 then this also enables NOP generation unless the '-mN' is also specified.

'-msilicon-errata=NAME[,NAME...]'

Implements a fixup for named silicon errata. Multiple silicon errata can be specified by multiple uses of the '-msilicon-errata' option and/or by including the errata names, separated by commas, on an individual '-msilicon-errata' option. Errata names currently recognised by the assembler are:

'cpu4'

'PUSH #4' and 'PUSH #8' need longer encodings on the MSP430. This option is enabled by default, and cannot be disabled.

'cpu8'

Do not set the 'SP' to an odd value.

'cpu11'

Do not update the 'SR' and the 'PC' in the same instruction.

'cpu12'

Do not use the 'PC' in a 'CMP' or 'BIT' instruction.

'cpu13'

Do not use an arithmetic instruction to modify the 'SR'.

'cpu19'

Insert 'NOP' after 'CPUOFF'.

'-msilicon-errata-warn=NAME[,NAME...]'

Like the '-msilicon-errata' option except that instead of fixing the specified errata, a warning message is issued instead. This option can be used alongside '-msilicon-errata' to generate messages whenever a problem is fixed, or on its own in order to inspect code for potential problems.

'-mP'

enables polymorph instructions handler.

'-mQ'

enables relaxation at assembly time. DANGEROUS!

'-ml'

indicates that the input uses the large code model.

'-mn'

enables the generation of a NOP instruction following any instruction that might change the interrupts enabled/disabled state. The pipelined nature of the MSP430 core means that any instruction that changes the interrupt state ('EINT', 'DINT', 'BIC #8, SR', 'BIS #8, SR' or 'MOV.W <>, SR') must be followed by a NOP instruction in order to ensure the correct processing of interrupts. By default it is up to the programmer to supply these NOP instructions, but this command line option enables the automatic insertion by the assembler, if they are missing.

'-mN'

disables the generation of a NOP instruction following any instruction that might change the interrupts enabled/disabled

state. This is the default behaviour.

'-my'

tells the assembler to generate a warning message if a NOP does not immediately follow an instruction that enables or disables interrupts. This is the default.

Note that this option can be stacked with the '-mn' option so that the assembler will both warn about missing NOP instructions and then insert them automatically.

'-mY'

disables warnings about missing NOP instructions.

'-md'

mark the object file as one that requires data to be copied from ROM to RAM at execution startup. Disabled by default.

us

File: as.info, Node: MSP430 Syntax, Next: MSP430 Floating Point, Prev: MSP430 Options, Up: MSP430-Dependent

9.29.2 Syntax

* Menu:

```
* MSP430-Macros::      Macros
* MSP430-Chars::      Special Characters
* MSP430-Regs::       Register Names
* MSP430-Ext::        Assembler Extensions
```

us

File: as.info, Node: MSP430-Macros, Next: MSP430-Chars, Up: MSP430 Syntax

9.29.2.1 Macros

.....

The macro syntax used on the MSP 430 is like that described in the MSP 430 Family Assembler Specification. Normal 'as' macros should still work.

Additional built-in macros are:

'llo(exp)'

Extracts least significant word from 32-bit expression 'exp'.

'lhi(exp)'

Extracts most significant word from 32-bit expression 'exp'.

'hlo(exp)'

Extracts 3rd word from 64-bit expression 'exp'.

'hhi(exp)'

Extracts 4th word from 64-bit expression 'exp'.

They normally being used as an immediate source operand.

```
mov    #llo(1), r10    ; == mov #1, r10
mov    #lhi(1), r10    ; == mov #0, r10
```



File: as.info, Node: MSP430-Chars, Next: MSP430-Regs, Prev: MSP430-Macros, Up: MSP430 Syntax

9.29.2.2 Special Characters

.....

A semicolon (';') appearing anywhere on a line starts a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but it can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

Multiple statements can appear on the same line provided that they are separated by the '{' character.

The character '\$' in jump instructions indicates current location and implemented only for TI syntax compatibility.



File: as.info, Node: MSP430-Regs, Next: MSP430-Ext, Prev: MSP430-Chars, Up: MSP430 Syntax

9.29.2.3 Register Names

.....

General-purpose registers are represented by predefined symbols of the form 'rN' (for global registers), where N represents a number between '0' and '15'. The leading letters may be in either upper or lower case; for example, 'r13' and 'R7' are both valid register names.

Register names 'PC', 'SP' and 'SR' cannot be used as register names and will be treated as variables. Use 'r0', 'r1', and 'r2' instead.



File: as.info, Node: MSP430-Ext, Prev: MSP430-Regs, Up: MSP430 Syntax

9.29.2.4 Assembler Extensions

.....

'@rN'

As destination operand being treated as '0(rn)'

'0(rN)'

As source operand being treated as '@rn'

'jCOND +N'

Skips next N bytes followed by jump instruction and equivalent to 'jCOND \$+N+2'

Also, there are some instructions, which cannot be found in other assemblers. These are branch instructions, which has different opcodes upon jump distance. They all got PC relative addressing mode.

'beq label'

A polymorph instruction which is 'jeq label' in case if jump

distance within allowed range for cpu's jump instruction. If not, this unrolls into a sequence of

```
    jne $+6
    br label
```

'bne label'

A polymorph instruction which is 'jne label' or 'jeq +4; br label'

'blt label'

A polymorph instruction which is 'jl label' or 'jge +4; br label'

'bltn label'

A polymorph instruction which is 'jn label' or 'jn +2; jmp +4; br label'

'bltu label'

A polymorph instruction which is 'jlo label' or 'jhs +2; br label'

'bge label'

A polymorph instruction which is 'jge label' or 'jl +4; br label'

'bgeu label'

A polymorph instruction which is 'jhs label' or 'jlo +4; br label'

'bgt label'

A polymorph instruction which is 'jeq +2; jge label' or 'jeq +6; jl +4; br label'

'bgtu label'

A polymorph instruction which is 'jeq +2; jhs label' or 'jeq +6; jlo +4; br label'

'bleu label'

A polymorph instruction which is 'jeq label; jlo label' or 'jeq +2; jhs +4; br label'

'ble label'

A polymorph instruction which is 'jeq label; jl label' or 'jeq +2; jge +4; br label'

'jump label'

A polymorph instruction which is 'jmp label' or 'br label'



File: as.info, Node: MSP430 Floating Point, Next: MSP430 Directives, Prev: MSP430 Syntax, Up: MSP430-Dependent

9.29.3 Floating Point

The MSP 430 family uses IEEE 32-bit floating-point numbers.



File: as.info, Node: MSP430 Directives, Next: MSP430 Opcodes, Prev: MSP430 Floating Point, Up: MSP430-Dependent

9.29.4 MSP 430 Machine Directives

`.file`

This directive is ignored; it is accepted for compatibility with other MSP 430 assemblers.

`_Warning:` in other versions of the GNU assembler, `.file` is used for the directive called `.app-file` in the MSP 430 support.

`.line`

This directive is ignored; it is accepted for compatibility with other MSP 430 assemblers.

`.arch`

Sets the target microcontroller in the same way as the `-mmcu` command line option.

`.cpu`

Sets the target architecture in the same way as the `-mcpu` command line option.

`.profiler`

This directive instructs assembler to add new profile entry to the object file.

`.refsym`

This directive instructs assembler to add an undefined reference to the symbol following the directive. The maximum symbol name length is 1023 characters. No relocation is created for this symbol; it will exist purely for pulling in object files from archives. Note that this reloc is not sufficient to prevent garbage collection; use a `KEEP()` directive in the linker file to preserve such objects.



File: as.info, Node: MSP430 Opcodes, Next: MSP430 Profiling Capability, Prev: MSP430 Directives, Up: MSP430-Dependent

9.29.5 Opcodes

`'as'` implements all the standard MSP 430 opcodes. No additional pseudo-instructions are needed on this family.

For information on the 430 machine instruction set, see 'MSP430 User's Manual, document slau049d', Texas Instrument, Inc.



File: as.info, Node: MSP430 Profiling Capability, Prev: MSP430 Opcodes, Up: MSP430-Dependent

9.29.6 Profiling Capability

It is a performance hit to use `gcc`'s profiling approach for this tiny target. Even more - `jtag` hardware facility does not perform any profiling functions. However we've got `gdb`'s built-in simulator where we can do anything.

We define new section `.profiler` which holds all profiling information. We define new pseudo operation `.profiler` which will

instruct assembler to add new profile entry to the object file. Profile should take place at the present address.

Pseudo operation format:

```
'.profiler flags,function_to_profile [, cycle_corrector, extra]'
```

where:

'flags' is a combination of the following characters:

```
's'      function entry
'x'      function exit
'i'      function is in init section
'f'      function is in fini section
'l'      library call
'c'      libc standard call
'd'      stack value demand
'I'      interrupt service routine
'p'      prologue start
'p'      prologue end
'E'      epilogue start
'e'      epilogue end
'j'      long jump / sjlj unwind
'a'      an arbitrary code fragment
't'      extra parameter saved (a constant value like frame size)
```

'function_to_profile'

a function address

'cycle_corrector'

a value which should be added to the cycle counter, zero if omitted.

'extra'

any extra parameter, zero if omitted.

For example:

```
.global fxx
.type fxx,@function
fxx:
.LFrameOffset_fxx=0x08
.profiler "scdP", fxx      ; function entry.
                        ; we also demand stack value to be saved
push r11
push r10
```

```

    push r9
    push r8
    .profiler "cdpt",fxx,0, .LFrameOffset_fxx ; check stack value at this point
                                           ; (this is a prologue end)
                                           ; note, that spare var filled with
                                           ; the farme size

    mov r15,r8
...
    .profiler cdE,fxx ; check stack
    pop r8
    pop r9
    pop r10
    pop r11
    .profiler xcde,fxx,3 ; exit adds 3 to the cycle counter
    ret ; cause 'ret' insn takes 3 cycles

```



File: as.info, Node: NDS32-Dependent, Next: NiosII-Dependent, Prev: MSP430-Dependent, Up: Machine Dependencies

9.30 NDS32 Dependent Features

=====

The NDS32 processors family includes high-performance and low-power 32-bit processors for high-end to low-end. GNU 'as' for NDS32 architectures supports NDS32 ISA version 3. For detail about NDS32 instruction set, please see the AndeStar ISA User Manual which is available at <http://www.andestech.com/en/index/index.htm>

* Menu:

```

* NDS32 Options::      Assembler options
* NDS32 Syntax::      High-level assembly macros

```



File: as.info, Node: NDS32 Options, Next: NDS32 Syntax, Up: NDS32-Dependent

9.30.1 NDS32 Options

The NDS32 configurations of GNU 'as' support these special options:

```

'-O1'
    Optimize for performance.

'-Os'
    Optimize for space.

'-EL'
    Produce little endian data output.

'-EB'
    Produce little endian data output.

'-mpic'
    Generate PIC.

'-mno-fp-as-gp-relax'
    Suppress fp-as-gp relaxation for this file.

```


'-mb2bb-relax'
Back-to-back branch optimization.

'-mno-all-relax'
Suppress all relaxation for this file.

'-march=<arch name>'
Assemble for architecture <arch name> which could be v3, v3j, v3m, v3f, v3s, v2, v2j, v2f, v2s.

'-mbaseline=<baseline>'
Assemble for baseline <baseline> which could be v2, v3, v3m.

'-mfpu-freg=FREG'
Specify a FPU configuration.
'0 8 SP / 4 DP registers'
'1 16 SP / 8 DP registers'
'2 32 SP / 16 DP registers'
'3 32 SP / 32 DP registers'

'-mabi=ABI'
Specify a abi version <abi> could be v1, v2, v2fp, v2fpp.

'-m[no-]mac'
Enable/Disable Multiply instructions support.

'-m[no-]div'
Enable/Disable Divide instructions support.

'-m[no-]16bit-ext'
Enable/Disable 16-bit extension

'-m[no-]dx-regs'
Enable/Disable d0/d1 registers

'-m[no-]perf-ext'
Enable/Disable Performance extension

'-m[no-]perf2-ext'
Enable/Disable Performance extension 2

'-m[no-]string-ext'
Enable/Disable String extension

'-m[no-]reduced-regs'
Enable/Disable Reduced Register configuration (GPR16) option

'-m[no-]audio-isa-ext'
Enable/Disable AUDIO ISA extension

'-m[no-]fpu-sp-ext'
Enable/Disable FPU SP extension

'-m[no-]fpu-dp-ext'
Enable/Disable FPU DP extension

'-m[no-]fpu-fma'
Enable/Disable FPU fused-multiply-add instructions

'-mall-ext'

Turn on all extensions and instructions support

US

File: as.info, Node: NDS32 Syntax, Prev: NDS32 Options, Up: NDS32-Dependent

9.30.2 Syntax

* Menu:

* NDS32-Chars:: Special Characters
 * NDS32-Regs:: Register Names
 * NDS32-Ops:: Pseudo Instructions

US

File: as.info, Node: NDS32-Chars, Next: NDS32-Regs, Up: NDS32 Syntax

9.30.2.1 Special Characters

.....

Use '#' at column 1 and '!' anywhere in the line except inside quotes.

Multiple instructions in a line are allowed though not recommended and should be separated by ';'.

Assembler is not case-sensitive in general except user defined label. For example, 'jral F1' is different from 'jral f1' while it is the same as 'JRAL F1'.

US

File: as.info, Node: NDS32-Regs, Next: NDS32-Ops, Prev: NDS32-Chars, Up: NDS32 Syntax

9.30.2.2 Register Names

.....

'General purpose registers (GPR)'

There are 32 32-bit general purpose registers \$r0 to \$r31.

'Accumulators d0 and d1'

64-bit accumulators: \$d0.hi, \$d0.lo, \$d1.hi, and \$d1.lo.

'Assembler reserved register \$ta'

Register \$ta (\$r15) is reserved for assembler using.

'Operating system reserved registers \$p0 and \$p1'

Registers \$p0 (\$r26) and \$p1 (\$r27) are used by operating system as scratch registers.

'Frame pointer \$fp'

Register \$r28 is regarded as the frame pointer.

'Global pointer'

Register \$r29 is regarded as the global pointer.

'Link pointer'

Register \$r30 is regarded as the link pointer.

'Stack pointer'
 Register \$r31 is regarded as the stack pointer.



File: as.info, Node: NDS32-Ops, Prev: NDS32-Regs, Up: NDS32 Syntax

9.30.2.3 Pseudo Instructions

.....

'li rt5,imm32'
 load 32-bit integer into register rt5. 'sethi rt5,hi20(imm32)' and then 'ori rt5,reg,lo12(imm32)'.

'la rt5,var'
 Load 32-bit address of var into register rt5. 'sethi rt5,hi20(var)' and then 'ori reg,rt5,lo12(var)'

'l.[bhw] rt5,var'
 Load value of var into register rt5. 'sethi \$ta,hi20(var)' and then 'l[bhw]i rt5,[\$ta+lo12(var)]'

'l.[bh]s rt5,var'
 Load value of var into register rt5. 'sethi \$ta,hi20(var)' and then 'l[bh]si rt5,[\$ta+lo12(var)]'

'l.[bhw]p rt5,var,inc'
 Load value of var into register rt5 and increment \$ta by amount inc. 'la \$ta,var' and then 'l[bhw]i.bi rt5,[\$ta],inc'

'l.[bhw]pc rt5,inc'
 Continue loading value of var into register rt5 and increment \$ta by amount inc. 'l[bhw]i.bi rt5,[\$ta],inc.'

'l.[bh]sp rt5,var,inc'
 Load value of var into register rt5 and increment \$ta by amount inc. 'la \$ta,var' and then 'l[bh]si.bi rt5,[\$ta],inc'

'l.[bh]spc rt5,inc'
 Continue loading value of var into register rt5 and increment \$ta by amount inc. 'l[bh]si.bi rt5,[\$ta],inc.'

's.[bhw] rt5,var'
 Store register rt5 to var. 'sethi \$ta,hi20(var)' and then 's[bhw]i rt5,[\$ta+lo12(var)]'

's.[bhw]p rt5,var,inc'
 Store register rt5 to var and increment \$ta by amount inc. 'la \$ta,var' and then 's[bhw]i.bi rt5,[\$ta],inc'

's.[bhw]pc rt5,inc'
 Continue storing register rt5 to var and increment \$ta by amount inc. 's[bhw]i.bi rt5,[\$ta],inc.'

'not rt5,ra5'
 Alias of 'nor rt5,ra5,ra5'.

'neg rt5,ra5'
 Alias of 'subri rt5,ra5,0'.

'br rb5'
Depending on how it is assembled, it is translated into 'r5 rb5' or 'jr rb5'.

'b label'
Branch to label depending on how it is assembled, it is translated into 'j8 label', 'j label', or "'la \$ta,label' 'br \$ta'".

'bral rb5'
Alias of jral br5 depending on how it is assembled, it is translated into 'jral5 rb5' or 'jral rb5'.

'bal fname'
Alias of jal fname depending on how it is assembled, it is translated into 'jal fname' or "'la \$ta,fname' 'bral \$ta'".

'call fname'
Call function fname same as 'jal fname'.

'move rt5,ra5'
For 16-bit, this is 'mov55 rt5,ra5'. For no 16-bit, this is 'ori rt5,ra5,0'.

'move rt5,var'
This is the same as 'l.w rt5,var'.

'move rt5,imm32'
This is the same as 'li rt5,imm32'.

'pushm ra5,rb5'
Push contents of registers from ra5 to rb5 into stack.

'push ra5'
Push content of register ra5 into stack. (same 'pushm ra5,ra5').

'push.d var'
Push value of double-word variable var into stack.

'push.w var'
Push value of word variable var into stack.

'push.h var'
Push value of half-word variable var into stack.

'push.b var'
Push value of byte variable var into stack.

'pusha var'
Push 32-bit address of variable var into stack.

'pushi imm32'
Push 32-bit immediate value into stack.

'popm ra5,rb5'
Pop top of stack values into registers ra5 to rb5.

'pop rt5'
Pop top of stack value into register. (same as 'popm rt5,rt5'.)

'pop.d var,ra5'
 Pop value of double-word variable var from stack using register ra5
 as 2nd scratch register. (1st is \$ta)

'pop.w var,ra5'
 Pop value of word variable var from stack using register ra5.

'pop.h var,ra5'
 Pop value of half-word variable var from stack using register ra5.

'pop.b var,ra5'
 Pop value of byte variable var from stack using register ra5.



File: as.info, Node: NiosII-Dependent, Next: NS32K-Dependent, Prev: NDS32-Dependent, Up: Machine Dependencies

9.31 Nios II Dependent Features

=====

* Menu:

* Nios II Options::	Options
* Nios II Syntax::	Syntax
* Nios II Relocations::	Relocations
* Nios II Directives::	Nios II Machine Directives
* Nios II Opcodes::	Opcodes



File: as.info, Node: Nios II Options, Next: Nios II Syntax, Up: NiosII-Dependent

9.31.1 Options

'-relax-section'

Replace identified out-of-range branches with PC-relative 'jmp' sequences when possible. The generated code sequences are suitable for use in position-independent code, but there is a practical limit on the extended branch range because of the length of the sequences. This option is the default.

'-relax-all'

Replace branch instructions not determinable to be in range and all call instructions with 'jmp' and 'callr' sequences (respectively). This option generates absolute relocations against the target symbols and is not appropriate for position-independent code.

'-no-relax'

Do not replace any branches or calls.

'-EB'

Generate big-endian output.

'-EL'

Generate little-endian output. This is the default.

'-march=ARCHITECTURE'

This option specifies the target architecture. The assembler


```
.int 123
.section .text
ldw r4, %gprel(fastint)(gp)
```

```
'%call(EXPRESSION)'
'%call_lo(EXPRESSION)'
'%call_hiadj(EXPRESSION)'
'%got(EXPRESSION)'
'%got_lo(EXPRESSION)'
'%got_hiadj(EXPRESSION)'
'%gotoff(EXPRESSION)'
'%gotoff_lo(EXPRESSION)'
'%gotoff_hiadj(EXPRESSION)'
'%tls_gd(EXPRESSION)'
'%tls_ie(EXPRESSION)'
'%tls_le(EXPRESSION)'
'%tls_ldm(EXPRESSION)'
'%tls_ldo(EXPRESSION)'
```

These relocations support the ABI for Linux Systems documented in the 'Nios II Processor Reference Handbook'.



File: as.info, Node: Nios II Directives, Next: Nios II Opcodes, Prev: Nios II Relocations, Up: NiosII-Dependent

9.31.4 Nios II Machine Directives

```
-----
'.align EXPRESSION [, EXPRESSION]'
    This is the generic '.align' directive, however this aligns to a
    power of two.

'.half EXPRESSION'
    Create an aligned constant 2 bytes in size.

'.word EXPRESSION'
    Create an aligned constant 4 bytes in size.

'.dword EXPRESSION'
    Create an aligned constant 8 bytes in size.

'.2byte EXPRESSION'
    Create an unaligned constant 2 bytes in size.

'.4byte EXPRESSION'
    Create an unaligned constant 4 bytes in size.

'.8byte EXPRESSION'
    Create an unaligned constant 8 bytes in size.

'.16byte EXPRESSION'
    Create an unaligned constant 16 bytes in size.

'.set noat'
    Allows assembly code to use 'at' register without warning. Macro
    or relaxation expansions generate warnings.

'.set at'
```


9.32.1.1 Special Characters

.....

The presence of a '#' appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

If Sequent compatibility has been configured into the assembler then the '|' character appearing as the first character on a line will also indicate the start of a line comment.

The ';' character can be used to separate statements on the same line.



File: as.info, Node: PDP-11-Dependent, Next: PJ-Dependent, Prev: NS32K-Dependent, Up: Machine Dependencies

9.33 PDP-11 Dependent Features

=====

* Menu:

- * PDP-11-Options:: Options
- * PDP-11-Pseudos:: Assembler Directives
- * PDP-11-Syntax:: DEC Syntax versus BSD Syntax
- * PDP-11-Mnemonics:: Instruction Naming
- * PDP-11-Synthetic:: Synthetic Instructions



File: as.info, Node: PDP-11-Options, Next: PDP-11-Pseudos, Up: PDP-11-Dependent

9.33.1 Options

The PDP-11 version of 'as' has a rich set of machine dependent options.

9.33.1.1 Code Generation Options

.....

'-mpic | -mno-pic'
Generate position-independent (or position-dependent) code.

The default is to generate position-independent code.

9.33.1.2 Instruction Set Extension Options

.....

These options enables or disables the use of extensions over the base line instruction set as introduced by the first PDP-11 CPU: the KA11. Most options come in two variants: a '-m'EXTENSION that enables EXTENSION, and a '-mno-'EXTENSION that disables EXTENSION.

The default is to enable all extensions.

- '-mall | -mall-extensions'
Enable all instruction set extensions.
- '-mno-extensions'
Disable all instruction set extensions.
- '-mcis | -mno-cis'
Enable (or disable) the use of the commercial instruction set, which consists of these instructions: 'ADDNI', 'ADDN', 'ADDPI', 'ADDP', 'ASHNI', 'ASHN', 'ASHPI', 'ASHP', 'CMPCI', 'CMPC', 'CMPNI', 'CMPN', 'CMPPI', 'CMPP', 'CVTLNI', 'CVTLN', 'CVTLPI', 'CVTLP', 'CVTNLI', 'CVTNL', 'CVTNPI', 'CVTNP', 'CVTPLI', 'CVTPL', 'CVTPNI', 'CVTPN', 'DIVPI', 'DIVP', 'L2DR', 'L3DR', 'LOCCI', 'LOCC', 'MATCI', 'MATC', 'MOVCI', 'MOVC', 'MOVRCI', 'MOVRC', 'MOVTCI', 'MOVTC', 'MULPI', 'MULP', 'SCANCI', 'SCANC', 'SKPCI', 'SKPC', 'SPANCI', 'SPANC', 'SUBNI', 'SUBN', 'SUBPI', and 'SUBP'.
- '-mcs | -mno-csm'
Enable (or disable) the use of the 'CSM' instruction.
- '-meis | -mno-eis'
Enable (or disable) the use of the extended instruction set, which consists of these instructions: 'ASHC', 'ASH', 'DIV', 'MARK', 'MUL', 'RTT', 'SOB', 'SXT', and 'XOR'.
- '-mfis | -mkev11'
'-mno-fis | -mno-kev11'
Enable (or disable) the use of the KEV11 floating-point instructions: 'FADD', 'FDIV', 'FMUL', and 'FSUB'.
- '-mfpp | -mfpu | -mfp-11'
'-mno-fpp | -mno-fpu | -mno-fp-11'
Enable (or disable) the use of FP-11 floating-point instructions: 'ABSF', 'ADDF', 'CFCC', 'CLRF', 'CMPF', 'DIVF', 'LDCFF', 'LDCIF', 'LDEXP', 'LDF', 'LDFPS', 'MODF', 'MULF', 'NEGF', 'SETD', 'SETF', 'SETI', 'SETL', 'STCFF', 'STCFI', 'STEXP', 'STF', 'STFPS', 'STST', 'SUBF', and 'TSTF'.
- '-mlimited-eis | -mno-limited-eis'
Enable (or disable) the use of the limited extended instruction set: 'MARK', 'RTT', 'SOB', 'SXT', and 'XOR'.
- The -mno-limited-eis options also implies -mno-eis.
- '-mmfpt | -mno-mfpt'
Enable (or disable) the use of the 'MFPT' instruction.
- '-mmultiproc | -mno-multiproc'
Enable (or disable) the use of multiprocessor instructions: 'TSTSET' and 'WRTLCK'.
- '-mmxps | -mno-mxps'
Enable (or disable) the use of the 'MFPS' and 'MTPS' instructions.
- '-mspl | -mno-spl'
Enable (or disable) the use of the 'SPL' instruction.
- Enable (or disable) the use of the microcode instructions: 'LDUB', 'MED', and 'XFC'.

9.33.1.3 CPU Model Options

.....

These options enable the instruction set extensions supported by a particular CPU, and disables all other extensions.

- '-mka11'
KA11 CPU. Base line instruction set only.
- '-mkb11'
KB11 CPU. Enable extended instruction set and 'SPL'.
- '-mkd11a'
KD11-A CPU. Enable limited extended instruction set.
- '-mkd11b'
KD11-B CPU. Base line instruction set only.
- '-mkd11d'
KD11-D CPU. Base line instruction set only.
- '-mkd11e'
KD11-E CPU. Enable extended instruction set, 'MFPS', and 'MTPS'.
- '-mkd11f | -mkd11h | -mkd11q'
KD11-F, KD11-H, or KD11-Q CPU. Enable limited extended instruction set, 'MFPS', and 'MTPS'.
- '-mkd11k'
KD11-K CPU. Enable extended instruction set, 'LDUB', 'MED', 'MFPS', 'MFPT', 'MTPS', and 'XFC'.
- '-mkd11z'
KD11-Z CPU. Enable extended instruction set, 'CSM', 'MFPS', 'MFPT', 'MTPS', and 'SPL'.
- '-mf11'
F11 CPU. Enable extended instruction set, 'MFPS', 'MFPT', and 'MTPS'.
- '-mj11'
J11 CPU. Enable extended instruction set, 'CSM', 'MFPS', 'MFPT', 'MTPS', 'SPL', 'TSTSET', and 'WRTLCK'.
- '-mt11'
T11 CPU. Enable limited extended instruction set, 'MFPS', and 'MTPS'.

9.33.1.4 Machine Model Options

.....

These options enable the instruction set extensions supported by a particular machine model, and disables all other extensions.

- '-m11/03'
Same as '-mkd11f'.
- '-m11/04'

Same as '-mkd11d'.

'-m11/05 | -m11/10'
Same as '-mkd11b'.

'-m11/15 | -m11/20'
Same as '-mka11'.

'-m11/21'
Same as '-mt11'.

'-m11/23 | -m11/24'
Same as '-mf11'.

'-m11/34'
Same as '-mkd11e'.

'-m11/34a'
Ame as '-mkd11e' '-mfpp'.

'-m11/35 | -m11/40'
Same as '-mkd11a'.

'-m11/44'
Same as '-mkd11z'.

'-m11/45 | -m11/50 | -m11/55 | -m11/70'
Same as '-mkb11'.

'-m11/53 | -m11/73 | -m11/83 | -m11/84 | -m11/93 | -m11/94'
Same as '-mj11'.

'-m11/60'
Same as '-mkd11k'.



File: as.info, Node: PDP-11-Pseudos, Next: PDP-11-Syntax, Prev: PDP-11-Options,
Up: PDP-11-Dependent

9.33.2 Assembler Directives

The PDP-11 version of 'as' has a few machine dependent assembler directives.

'**.bss**'
Switch to the 'bss' section.

'**.even**'
Align the location counter to an even number.



File: as.info, Node: PDP-11-Syntax, Next: PDP-11-Mnemonics, Prev: PDP-11-Pseudos,
Up: PDP-11-Dependent

9.33.3 PDP-11 Assembly Language Syntax

'as' supports both DEC syntax and BSD syntax. The only difference is

that in DEC syntax, a '#' character is used to denote an immediate constants, while in BSD syntax the character for this purpose is '\$'.

general-purpose registers are named 'r0' through 'r7'. Mnemonic alternatives for 'r6' and 'r7' are 'sp' and 'pc', respectively.

Floating-point registers are named 'ac0' through 'ac3', or alternatively 'fr0' through 'fr3'.

Comments are started with a '#' or a '/' character, and extend to the end of the line. (FIXME: clash with immediates?)

Multiple statements on the same line can be separated by the ';' character.



File: as.info, Node: PDP-11-Mnemonics, Next: PDP-11-Synthetic, Prev: PDP-11-Syntax, Up: PDP-11-Dependent

9.33.4 Instruction Naming

Some instructions have alternative names.

```
'BCC'
    'BHIS'

'BCS'
    'BLO'

'L2DR'
    'L2D'

'L3DR'
    'L3D'

'SYS'
    'TRAP'
```



File: as.info, Node: PDP-11-Synthetic, Prev: PDP-11-Mnemonics, Up: PDP-11-Dependent

9.33.5 Synthetic Instructions

The 'JBR' and 'J'CC synthetic instructions are not supported yet.



File: as.info, Node: PJ-Dependent, Next: PPC-Dependent, Prev: PDP-11-Dependent, Up: Machine Dependencies

9.34 picoJava Dependent Features

=====

* Menu:

```
* PJ Options::          Options
* PJ Syntax::          PJ Syntax
```



File: as.info, Node: PJ Options, Next: PJ Syntax, Up: PJ-Dependent

9.34.1 Options

'as' has two additional command-line options for the picoJava architecture.

'-ml'

This option selects little endian data output.

'-mb'

This option selects big endian data output.



File: as.info, Node: PJ Syntax, Prev: PJ Options, Up: PJ-Dependent

9.34.2 PJ Syntax

* Menu:

* PJ-Chars:: Special Characters



File: as.info, Node: PJ-Chars, Up: PJ Syntax

9.34.2.1 Special Characters

.....

The presence of a '!' or '/' on a line indicates the start of a comment that extends to the end of the current line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

The ';' character can be used to separate statements on the same line.



File: as.info, Node: PPC-Dependent, Next: RL78-Dependent, Prev: PJ-Dependent, Up: Machine Dependencies

9.35 PowerPC Dependent Features

=====

* Menu:

* PowerPC-Opts:: Options
 * PowerPC-Pseudo:: PowerPC Assembler Directives
 * PowerPC-Syntax:: PowerPC Syntax



File: as.info, Node: PowerPC-Opts, Next: PowerPC-Pseudo, Up: PPC-Dependent

9.35.1 Options

The PowerPC chip family includes several successive levels, using the same core instruction set, but including a few additional instructions at each level. There are exceptions to this however. For details on what instructions each variant supports, please see the chip's architecture reference manual.

The following table lists all available PowerPC options.

- '-a32'
Generate ELF32 or XCOFF32.
- '-a64'
Generate ELF64 or XCOFF64.
- '-K PIC'
Set EF_PPC_RELOCATABLE_LIB in ELF flags.
- '-mpwrx | -mpwr2'
Generate code for POWER/2 (RIOS2).
- '-mpwr'
Generate code for POWER (RIOS1)
- '-m601'
Generate code for PowerPC 601.
- '-mppc, -mppc32, -m603, -m604'
Generate code for PowerPC 603/604.
- '-m403, -m405'
Generate code for PowerPC 403/405.
- '-m440'
Generate code for PowerPC 440. BookE and some 405 instructions.
- '-m464'
Generate code for PowerPC 464.
- '-m476'
Generate code for PowerPC 476.
- '-m7400, -m7410, -m7450, -m7455'
Generate code for PowerPC 7400/7410/7450/7455.
- '-m750cl'
Generate code for PowerPC 750CL.
- '-m821, -m850, -m860'
Generate code for PowerPC 821/850/860.
- '-mppc64, -m620'
Generate code for PowerPC 620/625/630.
- '-me500, -me500x2'
Generate code for Motorola e500 core complex.
- '-me500mc'

Generate code for Freescale e500mc core complex.

'-me500mc64'

Generate code for Freescale e500mc64 core complex.

'-me5500'

Generate code for Freescale e5500 core complex.

'-me6500'

Generate code for Freescale e6500 core complex.

'-mspe'

Generate code for Motorola SPE instructions.

'-mtitan'

Generate code for AppliedMicro Titan core complex.

'-mppc64bridge'

Generate code for PowerPC 64, including bridge insns.

'-mbooke'

Generate code for 32-bit BookE.

'-ma2'

Generate code for A2 architecture.

'-me300'

Generate code for PowerPC e300 family.

'-maltivec'

Generate code for processors with AltiVec instructions.

'-mvle'

Generate code for Freescale PowerPC VLE instructions.

'-mvsx'

Generate code for processors with Vector-Scalar (VSX) instructions.

'-mhtm'

Generate code for processors with Hardware Transactional Memory instructions.

'-mpower4, -mpwr4'

Generate code for Power4 architecture.

'-mpower5, -mpwr5, -mpwr5x'

Generate code for Power5 architecture.

'-mpower6, -mpwr6'

Generate code for Power6 architecture.

'-mpower7, -mpwr7'

Generate code for Power7 architecture.

'-mpower8, -mpwr8'

Generate code for Power8 architecture.

'-mpower9, -mpwr9'

Generate code for Power9 architecture.

'-mcell'
'-mcell'
 Generate code for Cell Broadband Engine architecture.

'-mcom'
 Generate code Power/PowerPC common instructions.

'-many'
 Generate code for any architecture (PWR/PWRX/PPC).

'-mregnames'
 Allow symbolic names for registers.

'-mno-regnames'
 Do not allow symbolic names for registers.

'-mrelocatable'
 Support for GCC's -mrelocatable option.

'-mrelocatable-lib'
 Support for GCC's -mrelocatable-lib option.

'-memb'
 Set PPC_EMB bit in ELF flags.

'-mlittle, -mlittle-endian, -le'
 Generate code for a little endian machine.

'-mbig, -mbig-endian, -be'
 Generate code for a big endian machine.

'-msolaris'
 Generate code for Solaris.

'-mno-solaris'
 Do not generate code for Solaris.

'-nops=COUNT'
 If an alignment directive inserts more than COUNT nops, put a
 branch at the beginning to skip execution of the nops.



File: as.info, Node: PowerPC-Pseudo, Next: PowerPC-Syntax, Prev: PowerPC-Opts,
Up: PPC-Dependent

9.35.2 PowerPC Assembler Directives

A number of assembler directives are available for PowerPC. The following table is far from complete.

'**.machine "string"**'
 This directive allows you to change the machine for which code is generated. '**"string"**' may be any of the -m cpu selection options (without the -m) enclosed in double quotes, '**"push"**', or '**"pop"**'. '**.machine "push"**' saves the currently selected cpu, which may be restored with '**.machine "pop"**'.

architecture.

'mg13'
Mark the generated binary as targeting the G13 variant of the RL78 architecture.

'mg14'
'mrl78'
Mark the generated binary as targeting the G14 variant of the RL78 architecture. This is the default.

'm32bit-doubles'
Mark the generated binary as one that uses 32-bits to hold the 'double' floating point type. This is the default.

'm64bit-doubles'
Mark the generated binary as one that uses 64-bits to hold the 'double' floating point type.



File: as.info, Node: RL78-Modifiers, Next: RL78-Directives, Prev: RL78-Opts, Up: RL78-Dependent

9.36.2 Symbolic Operand Modifiers

The RL78 has three modifiers that adjust the relocations used by the linker:

'%lo16()'

When loading a 20-bit (or wider) address into registers, this modifier selects the 16 least significant bits.

```
movw ax, #%lo16(_sym)
```

'%hi16()'

When loading a 20-bit (or wider) address into registers, this modifier selects the 16 most significant bits.

```
movw ax, #%hi16(_sym)
```

'%hi8()'

When loading a 20-bit (or wider) address into registers, this modifier selects the 8 bits that would go into CS or ES (i.e. bits 23..16).

```
mov es, #%hi8(_sym)
```



File: as.info, Node: RL78-Directives, Next: RL78-Syntax, Prev: RL78-Modifiers, Up: RL78-Dependent

9.36.3 Assembler Directives

In addition to the common directives, the RL78 adds these:

`double`
 Output a constant in "double" format, which is either a 32-bit or a 64-bit floating point value, depending upon the setting of the `-m32bit-doubles` or `-m64bit-doubles` command line option.

`bss`
 Select the BSS section.

`3byte`
 Output a constant value in a three byte format.

`int`
`word`
 Output a constant value in a four byte format.



File: as.info, Node: RL78-Syntax, Prev: RL78-Directives, Up: RL78-Dependent

9.36.4 Syntax for the RL78

* Menu:

* RL78-Chars:: Special Characters



File: as.info, Node: RL78-Chars, Up: RL78-Syntax

9.36.4.1 Special Characters

.....

The presence of a `';` appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a `'#'` appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (`*note Comments::`) or a preprocessor control command (`*note Preprocessing::`).

The `'|'` character can be used to separate statements on the same line.



File: as.info, Node: RISC-V-Dependent, Next: RX-Dependent, Prev: RL78-Dependent, Up: Machine Dependencies

9.37 RISC-V Dependent Features

=====

* Menu:

* RISC-V-Opts:: RISC-V Options



File: as.info, Node: RISC-V-Opts, Up: RISC-V-Dependent

9.37.1 Options

The following table lists all available RISC-V specific options

- '-fpic'
Generate position-independent code
- '-fno-pic'
Don't generate position-independent code (default)
- '-march=ISA'
Select the base isa, as specified by ISA. For example
-march=rv32ima.
- '-mabi=ABI'
Selects the ABI, which is either "ilp32" or "lp64", optionally
followed by "f", "d", or "q" to indicate single-precision,
double-precision, or quad-precision floating-point calling
convention, or none to indicate the soft-float calling convention.



File: as.info, Node: RX-Dependent, Next: S/390-Dependent, Prev: RISC-V-Dependent,
Up: Machine Dependencies

9.38 RX Dependent Features

=====

* Menu:

- * RX-Opts:: RX Assembler Command Line Options
- * RX-Modifiers:: Symbolic Operand Modifiers
- * RX-Directives:: Assembler Directives
- * RX-Float:: Floating Point
- * RX-Syntax:: Syntax



File: as.info, Node: RX-Opts, Next: RX-Modifiers, Up: RX-Dependent

9.38.1 RX Options

The Renesas RX port of 'as' has a few target specific command line options:

- '-m32bit-doubles'
This option controls the ABI and indicates to use a 32-bit float ABI. It has no effect on the assembled instructions, but it does influence the behaviour of the '.double' pseudo-op. This is the default.
- '-m64bit-doubles'
This option controls the ABI and indicates to use a 64-bit float ABI. It has no effect on the assembled instructions, but it does influence the behaviour of the '.double' pseudo-op.
- '-mbig-endian'
This option controls the ABI and indicates to use a big-endian data ABI. It has no effect on the assembled instructions, but it does influence the behaviour of the '.short', '.hword', '.int', '.word', '.long', '.quad' and '.octa' pseudo-ops.

- '-mlittle-endian'
This option controls the ABI and indicates to use a little-endian data ABI. It has no effect on the assembled instructions, but it does influence the behaviour of the '.short', '.hword', '.int', '.word', '.long', '.quad' and '.octa' pseudo-ops. This is the default.
- '-muse-conventional-section-names'
This option controls the default names given to the code (.text), initialised data (.data) and uninitialised data sections (.bss).
- '-muse-renesas-section-names'
This option controls the default names given to the code (.P), initialised data (.D_1) and uninitialised data sections (.B_1). This is the default.
- '-msmall-data-limit'
This option tells the assembler that the small data limit feature of the RX port of GCC is being used. This results in the assembler generating an undefined reference to a symbol called '__gp' for use by the relocations that are needed to support the small data limit feature. This option is not enabled by default as it would otherwise pollute the symbol table.
- '-mpid'
This option tells the assembler that the position independent data of the RX port of GCC is being used. This results in the assembler generating an undefined reference to a symbol called '__pid_base', and also setting the RX_PID flag bit in the e_flags field of the ELF header of the object file.
- '-mint-register=NUM'
This option tells the assembler how many registers have been reserved for use by interrupt handlers. This is needed in order to compute the correct values for the '%gpreg' and '%pidreg' meta registers.
- '-mgcc-abi'
This option tells the assembler that the old GCC ABI is being used by the assembled code. With this version of the ABI function arguments that are passed on the stack are aligned to a 32-bit boundary.
- '-mrx-abi'
This option tells the assembler that the official RX ABI is being used by the assembled code. With this version of the ABI function arguments that are passed on the stack are aligned to their natural alignments. This option is the default.
- '-mcpu=NAME'
This option tells the assembler the target CPU type. Currently the 'rx100', 'rx200', 'rx600', 'rx610' and 'rxv2' are recognised as valid cpu names. Attempting to assemble an instruction not supported by the indicated cpu type will result in an error message being generated.
- '-mno-allow-string-insns'
This option tells the assembler to mark the object file that it is

building as one that does not use the string instructions 'SMOVF', 'SCMPU', 'SMOVB', 'SMOVU', 'SUNTIL' 'SWHILE' or the 'RMPA' instruction. In addition the mark tells the linker to complain if an attempt is made to link the binary with another one that does use any of these instructions.

Note - the inverse of this option, '-mallow-string-insns', is not needed. The assembler automatically detects the use of the the instructions in the source code and labels the resulting object file appropriately. If no string instructions are detected then the object file is labelled as being one that can be linked with either string-using or string-banned object files.



File: as.info, Node: RX-Modifiers, Next: RX-Directives, Prev: RX-Opts, Up: RX-Dependent

9.38.2 Symbolic Operand Modifiers

The assembler supports one modifier when using symbol addresses in RX instruction operands. The general syntax is the following:

`%gp(symbol)`

The modifier returns the offset from the `__GP` symbol to the specified symbol as a 16-bit value. The intent is that this offset should be used in a register+offset move instruction when generating references to small data. Ie, like this:

```
mov.W    %gp(_foo)[%gpreg], r1
```

The assembler also supports two meta register names which can be used to refer to registers whose values may not be known to the programmer. These meta register names are:

'%gpreg'

The small data address register.

'%pidreg'

The PID base address register.

Both registers normally have the value r13, but this can change if some registers have been reserved for use by interrupt handlers or if both the small data limit and position independent data features are being used at the same time.



File: as.info, Node: RX-Directives, Next: RX-Float, Prev: RX-Modifiers, Up: RX-Dependent

9.38.3 Assembler Directives

The RX version of 'as' has the following specific assembler directives:

'`.3byte`'

Inserts a 3-byte value into the output file at the current location.

`'.fetchalign'`

If the next opcode following this directive spans a fetch line boundary (8 byte boundary), the opcode is aligned to that boundary. If the next opcode does not span a fetch line, this directive has no effect. Note that one or more labels may be between this directive and the opcode; those labels are aligned as well. Any inserted bytes due to alignment will form a NOP opcode.



File: as.info, Node: RX-Float, Next: RX-Syntax, Prev: RX-Directives, Up: RX-Dependent

9.38.4 Floating Point

The floating point formats generated by directives are these.

`'.float'`

'Single' precision (32-bit) floating point constants.

`'.double'`

If the `'-m64bit-doubles'` command line option has been specified then then `'double'` directive generates `'double'` precision (64-bit) floating point constants, otherwise it generates `'single'` precision (32-bit) floating point constants. To force the generation of 64-bit floating point constants used the `'dc.d'` directive instead.



File: as.info, Node: RX-Syntax, Prev: RX-Float, Up: RX-Dependent

9.38.5 Syntax for the RX

* Menu:

* RX-Chars:: Special Characters



File: as.info, Node: RX-Chars, Up: RX-Syntax

9.38.5.1 Special Characters

The presence of a `'#'` appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a `'#'` appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (`*note Comments::`) or a preprocessor control command (`*note Preprocessing::`).

The `'!'` character can be used to separate statements on the same line.



File: as.info, Node: S/390-Dependent, Next: SCORE-Dependent, Prev: RX-Dependent, Up: Machine Dependencies

9.39 IBM S/390 Dependent Features

The s390 version of 'as' supports two architectures modes and eleven chip levels. The architecture modes are the Enterprise System Architecture (ESA) and the newer z/Architecture mode. The chip levels are g5 (or arch3), g6, z900 (or arch5), z990 (or arch6), z9-109, z9-ec (or arch7), z10 (or arch8), z196 (or arch9), zEC12 (or arch10), z13 (or arch11), and arch12.

* Menu:

- * s390 Options:: Command-line Options.
- * s390 Characters:: Special Characters.
- * s390 Syntax:: Assembler Instruction syntax.
- * s390 Directives:: Assembler Directives.
- * s390 Floating Point:: Floating Point.



File: as.info, Node: s390 Options, Next: s390 Characters, Up: S/390-Dependent

9.39.1 Options

The following table lists all available s390 specific options:

'-m31 | -m64'

Select 31- or 64-bit ABI implying a word size of 32- or 64-bit.

These options are only available with the ELF object file format, and require that the necessary BFD support has been included (on a 31-bit platform you must add `-enable-64-bit-bfd` on the call to the configure script to enable 64-bit usage and use s390x as target platform).

'-mesa | -mzarch'

Select the architecture mode, either the Enterprise System Architecture (esa) mode or the z/Architecture mode (zarch).

The 64-bit instructions are only available with the z/Architecture mode. The combination of `'-m64'` and `'-mesa'` results in a warning message.

'-march=CPU'

This option specifies the target processor. The following processor names are recognized: 'g5' (or 'arch3'), 'g6', 'z900' (or 'arch5'), 'z990' (or 'arch6'), 'z9-109', 'z9-ec' (or 'arch7'), 'z10' (or 'arch8'), 'z196' (or 'arch9'), 'zEC12' (or 'arch10') and 'z13' (or 'arch11').

Assembling an instruction that is not supported on the target processor results in an error message.

The processor names starting with 'arch' refer to the edition number in the Principle of Operations manual. They can be used as alternate processor names and have been added for compatibility with the IBM XL compiler.

'arch3', 'g5' and 'g6' cannot be used with the `'-mzarch'` option

since the z/Architecture mode is not supported on these processor levels.

There is no 'arch4' option supported. 'arch4' matches '-march=arch5 -mesa'.

'-mregnames'

Allow symbolic names for registers.

'-mno-regnames'

Do not allow symbolic names for registers.

'-mwarn-areg-zero'

Warn whenever the operand for a base or index register has been specified but evaluates to zero. This can indicate the misuse of general purpose register 0 as an address register.

US

File: as.info, Node: s390 Characters, Next: s390 Syntax, Prev: s390 Options, Up: S/390-Dependent

9.39.2 Special Characters

'#' is the line comment character.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The ';' character can be used instead of a newline to separate statements.

US

File: as.info, Node: s390 Syntax, Next: s390 Directives, Prev: s390 Characters, Up: S/390-Dependent

9.39.3 Instruction syntax

The assembler syntax closely follows the syntax outlined in Enterprise Systems Architecture/390 Principles of Operation (SA22-7201) and the z/Architecture Principles of Operation (SA22-7832).

Each instruction has two major parts, the instruction mnemonic and the instruction operands. The instruction format varies.

* Menu:

* s390 Register::	Register Naming
* s390 Mnemonics::	Instruction Mnemonics
* s390 Operands::	Instruction Operands
* s390 Formats::	Instruction Formats
* s390 Aliases::	Instruction Aliases
* s390 Operand Modifier::	Instruction Operand Modifier
* s390 Instruction Marker::	Instruction Marker
* s390 Literal Pool Entries::	Literal Pool Entries



File: as.info, Node: s390 Register, Next: s390 Mnemonics, Up: s390 Syntax

9.39.3.1 Register naming

.....

The 'as' recognizes a number of predefined symbols for the various processor registers. A register specification in one of the instruction formats is an unsigned integer between 0 and 15. The specific instruction and the position of the register in the instruction format denotes the type of the register. The register symbols are prefixed with '%':

```
%rN   the 16 general purpose registers, 0 <= N <= 15
%fN   the 16 floating point registers, 0 <= N <= 15
%aN   the 16 access registers, 0 <= N <= 15
%cN   the 16 control registers, 0 <= N <= 15
%lit  an alias for the general purpose register %r13
%sp   an alias for the general purpose register %r15
```



File: as.info, Node: s390 Mnemonics, Next: s390 Operands, Prev: s390 Register, Up: s390 Syntax

9.39.3.2 Instruction Mnemonics

.....

All instructions documented in the Principles of Operation are supported with the mnemonic and order of operands as described. The instruction mnemonic identifies the instruction format (*note s390 Formats:*) and the specific operation code for the instruction. For example, the 'lr' mnemonic denotes the instruction format 'RR' with the operation code '0x18'.

The definition of the various mnemonics follows a scheme, where the first character usually hint at the type of the instruction:

```
a      add instruction, for example 'al' for add logical 32-bit
b      branch instruction, for example 'bc' for branch on condition
c      compare or convert instruction, for example 'cr' for compare
       register 32-bit
d      divide instruction, for example 'dlr' devide logical register
       64-bit to 32-bit
i      insert instruction, for example 'ic' insert character
l      load instruction, for example 'ltr' load and test register
mv     move instruction, for example 'mvc' move character
m      multiply instruction, for example 'mh' multiply halfword
n      and instruction, for example 'ni' and immediate
o      or instruction, for example 'oc' or character
sla, sll  shift left single instruction
sra, srl  shift right single instruction
st      store instruction, for example 'stm' store multiple
s      subtract instruction, for example 'slr' subtract
       logical 32-bit
t      test or translate instruction, of example 'tm' test under mask
x      exclusive or instruction, for example 'xc' exclusive or
       character
```

Certain characters at the end of the mnemonic may describe a property

of the instruction:

```

c   the instruction uses a 8-bit character operand
f   the instruction extends a 32-bit operand to 64 bit
g   the operands are treated as 64-bit values
h   the operand uses a 16-bit halfword operand
i   the instruction uses an immediate operand
l   the instruction uses unsigned, logical operands
m   the instruction uses a mask or operates on multiple values
r   if r is the last character, the instruction operates on registers
y   the instruction uses 20-bit displacements

```

There are many exceptions to the scheme outlined in the above lists, in particular for the privileged instructions. For non-privileged instruction it works quite well, for example the instruction 'clgfr' c: compare instruction, l: unsigned operands, g: 64-bit operands, f: 32- to 64-bit extension, r: register operands. The instruction compares an 64-bit value in a register with the zero extended 32-bit value from a second register. For a complete list of all mnemonics see appendix B in the Principles of Operation.



File: as.info, Node: s390 Operands, Next: s390 Formats, Prev: s390 Mnemonics, Up: s390 Syntax

9.39.3.3 Instruction Operands

.....

Instruction operands can be grouped into three classes, operands located in registers, immediate operands, and operands in storage.

A register operand can be located in general, floating-point, access, or control register. The register is identified by a four-bit field. The field containing the register operand is called the R field.

Immediate operands are contained within the instruction and can have 8, 16 or 32 bits. The field containing the immediate operand is called the I field. Dependent on the instruction the I field is either signed or unsigned.

A storage operand consists of an address and a length. The address of a storage operands can be specified in any of these ways:

- * The content of a single general R
- * The sum of the content of a general register called the base register B plus the content of a displacement field D
- * The sum of the contents of two general registers called the index register X and the base register B plus the content of a displacement field
- * The sum of the current instruction address and a 32-bit signed immediate field multiplied by two.

The length of a storage operand can be:

- * Implied by the instruction
- * Specified by a bitmask
- * Specified by a four-bit or eight-bit length field L
- * Specified by the content of a general register

The notation for storage operand addresses formed from multiple fields is as follows:

'Dn(Bn)'

the address for operand number n is formed from the content of general register Bn called the base register and the displacement field Dn.

'Dn(Xn,Bn)'

the address for operand number n is formed from the content of general register Xn called the index register, general register Bn called the base register and the displacement field Dn.

'Dn(Ln,Bn)'

the address for operand number n is formed from the content of general register Bn called the base register and the displacement field Dn. The length of the operand n is specified by the field Ln.

The base registers Bn and the index registers Xn of a storage operand can be skipped. If Bn and Xn are skipped, a zero will be stored to the operand field. The notation changes as follows:

full notation	short notation
Dn(0,Bn)	Dn(Bn)
Dn(0,0)	Dn
Dn(0)	Dn
Dn(Ln,0)	Dn(Ln)



File: as.info, Node: s390 Formats, Next: s390 Aliases, Prev: s390 Operands, Up: s390 Syntax

9.39.3.4 Instruction Formats

.....

The Principles of Operation manuals lists 26 instruction formats where some of the formats have multiple variants. For the '.insn' pseudo directive the assembler recognizes some of the formats. Typically, the most general variant of the instruction format is used by the '.insn' directive.

The following table lists the abbreviations used in the table of instruction formats:

OpCode / OpCd	Part of the op code.
Bx	Base register number for operand x.
Dx	Displacement for operand x.
DLx	Displacement lower 12 bits for operand x.
DHx	Displacement higher 8-bits for operand x.
Rx	Register number for operand x.
Xx	Index register number for operand x.
Ix	Signed immediate for operand x.
Ux	Unsigned immediate for operand x.

An instruction is two, four, or six bytes in length and must be aligned on a 2 byte boundary. The first two bits of the instruction specify the length of the instruction, 00 indicates a two byte instruction, 01 and 10 indicates a four byte instruction, and 11 indicates a six byte instruction.

The following table lists the s390 instruction formats that are available with the '.insn' pseudo directive:

'E format'

```
+-----+
|   OpCode   |
+-----+
0             15
```

'RI format: <insn> R1,I2'

```
+-----+-----+-----+-----+
| OpCode | R1 | OpCd |      I2      |
+-----+-----+-----+-----+
0         8   12   16                   31
```

'RIE format: <insn> R1,R3,I2'

```
+-----+-----+-----+-----+-----+-----+
| OpCode | R1 | R3 |      I2      | |////////| OpCode |
+-----+-----+-----+-----+-----+-----+
0         8   12   16                   32       40       47
```

'RIL format: <insn> R1,I2'

```
+-----+-----+-----+-----+
| OpCode | R1 | OpCd |      I2      |
+-----+-----+-----+-----+
0         8   12   16                   47
```

'RILU format: <insn> R1,U2'

```
+-----+-----+-----+-----+
| OpCode | R1 | OpCd |      U2      |
+-----+-----+-----+-----+
0         8   12   16                   47
```

'RIS format: <insn> R1,I2,M3,D4(B4)'

```
+-----+-----+-----+-----+-----+-----+
| OpCode | R1 | M3 | B4 |      D4      | | I2 | Opcode |
+-----+-----+-----+-----+-----+-----+
0         8   12   16   20                   32       36       47
```

'RR format: <insn> R1,R2'

```
+-----+-----+-----+
| OpCode | R1 | R2 |
+-----+-----+-----+
0         8   12   15
```

'RRE format: <insn> R1,R2'

```
+-----+-----+-----+-----+
|   OpCode   | |////////| R1 | R2 |
+-----+-----+-----+-----+
0             16       24   28   31
```

'RRF format: <insn> R1,R2,R3,M4'

```
+-----+-----+-----+-----+-----+
|   OpCode   | R3 | M4 | R1 | R2 |
+-----+-----+-----+-----+-----+
0             16   20   24   28   31
```

'RRS format: <insn> R1,R2,M3,D4(B4)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  | B4  |      D4      | M3 |////| OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32   36   40    47

```

'RS format: <insn> R1,R3,D2(B2)'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  | B2  |      D2      |
+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                31

```

'RSE format: <insn> R1,R3,D2(B2)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  | B2  |      D2      |////////| OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32        40    47

```

'RSI format: <insn> R1,R3,I2'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  |      I2      |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16                                47

```

'RSY format: <insn> R1,R3,D2(B2)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  | B2  |      DL2     |      DH2     | OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32        40    47

```

'RX format: <insn> R1,D2(X2,B2)'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | X2  | B2  |      D2      |
+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                31

```

'RXE format: <insn> R1,D2(X2,B2)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | X2  | B2  |      D2      |////////| OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32        40    47

```

'RXF format: <insn> R1,R3,D2(X2,B2)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R3  | X2  | B2  |      D2      | R1 |///| OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32   36   40    47

```

'RXY format: <insn> R1,D2(X2,B2)'

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | X2  | B2  |      DL2     |      DH2     | OpCode |
+-----+-----+-----+-----+-----+-----+-----+-----+
0         8    12   16   20                32   36   40    47

```

'S format: <insn> D2(B2)'

```

+-----+-----+-----+-----+-----+-----+
|      OpCode      | B2  |      D2      |
+-----+-----+-----+-----+-----+-----+
0                   16   20                31

```

'SI format: <insn> D1(B1),I2'

```

+-----+-----+-----+-----+
| OpCode | I2  | B1  | D1  |
+-----+-----+-----+-----+
0        8      16  20      31

```

'SIY format: <insn> D1(B1),U2'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | I2  | B1  | DL1 | DH1 | OpCode |
+-----+-----+-----+-----+-----+-----+
0        8      16  20      32  36  40  47

```

'SIL format: <insn> D1(B1),I2'

```

+-----+-----+-----+-----+
| OpCode | B1  | D1  | I2  |
+-----+-----+-----+-----+
0        16  20      32      47

```

'SS format: <insn> D1(R1,B1),D2(B3),R3'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | R1  | R3  | B1  | D1  | B2  | D2  |
+-----+-----+-----+-----+-----+-----+
0        8   12  16  20      32  36      47

```

'SSE format: <insn> D1(B1),D2(B2)'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | B1  | D1  | B2  | D2  |
+-----+-----+-----+-----+-----+-----+
0        8   12  16  20      32  36      47

```

'SSF format: <insn> D1(B1),D2(B2),R3'

```

+-----+-----+-----+-----+-----+-----+
| OpCode | R3  | OpCd| B1  | D1  | B2  | D2  |
+-----+-----+-----+-----+-----+-----+
0        8   12  16  20      32  36      47

```

For the complete list of all instruction format variants see the Principles of Operation manuals.



File: as.info, Node: s390 Aliases, Next: s390 Operand Modifier, Prev: s390 Formats, Up: s390 Syntax

9.39.3.5 Instruction Aliases

.....

A specific bit pattern can have multiple mnemonics, for example the bit pattern '0xa7000000' has the mnemonics 'tmh' and 'tmlh'. In addition, there are a number of mnemonics recognized by 'as' that are not present in the Principles of Operation. These are the short forms of the branch instructions, where the condition code mask operand is encoded in the mnemonic. This is relevant for the branch instructions, the compare and branch instructions, and the compare and trap instructions.

For the branch instructions there are 20 condition code strings that can be used as part of the mnemonic in place of a mask operand in the instruction format:

instruction	short form
-------------	------------


```

-----
bcr    M1,R2          b<m>r   R2
bc     M1,D2(X2,B2)  b<m>    D2(X2,B2)
brc    M1,I2         j<m>    I2
brcl   M1,I2         jg<m>   I2

```

In the mnemonic for a branch instruction the condition code string <m> can be any of the following:

```

o      jump on overflow / if ones
h      jump on A high
p      jump on plus
nle    jump on not low or equal
l      jump on A low
m      jump on minus
nhe    jump on not high or equal
lh     jump on low or high
ne     jump on A not equal B
nz     jump on not zero / if not zeros
e      jump on A equal B
z      jump on zero / if zeroes
nlh    jump on not low or high
he     jump on high or equal
nl     jump on A not low
nm     jump on not minus / if not mixed
le     jump on low or equal
nh     jump on A not high
np     jump on not plus
no     jump on not overflow / if not ones

```

For the compare and branch, and compare and trap instructions there are 12 condition code strings that can be used as part of the mnemonic in place of a mask operand in the instruction format:

instruction	short form
crb R1,R2,M3,D4(B4)	crb<m> R1,R2,D4(B4)
cgrb R1,R2,M3,D4(B4)	cgrb<m> R1,R2,D4(B4)
crj R1,R2,M3,I4	crj<m> R1,R2,I4
cgrj R1,R2,M3,I4	cgrj<m> R1,R2,I4
cib R1,I2,M3,D4(B4)	cib<m> R1,I2,D4(B4)
cgib R1,I2,M3,D4(B4)	cgib<m> R1,I2,D4(B4)
cij R1,I2,M3,I4	cij<m> R1,I2,I4
cgij R1,I2,M3,I4	cgij<m> R1,I2,I4
crt R1,R2,M3	crt<m> R1,R2
cgrt R1,R2,M3	cgrt<m> R1,R2
cit R1,I2,M3	cit<m> R1,I2
cgit R1,I2,M3	cgit<m> R1,I2
clrb R1,R2,M3,D4(B4)	clrb<m> R1,R2,D4(B4)
clgrb R1,R2,M3,D4(B4)	clgrb<m> R1,R2,D4(B4)
clrj R1,R2,M3,I4	clrj<m> R1,R2,I4
clgrj R1,R2,M3,I4	clgrj<m> R1,R2,I4
clib R1,I2,M3,D4(B4)	clib<m> R1,I2,D4(B4)
clgib R1,I2,M3,D4(B4)	clgib<m> R1,I2,D4(B4)
clij R1,I2,M3,I4	clij<m> R1,I2,I4
clgij R1,I2,M3,I4	clgij<m> R1,I2,I4
clrt R1,R2,M3	clrt<m> R1,R2
clgrt R1,R2,M3	clgrt<m> R1,R2
clfit R1,I2,M3	clfit<m> R1,I2

```
clgit R1,I2,M3
```

```
clgit<m> R1,I2
```

In the mnemonic for a compare and branch and compare and trap instruction the condition code string <m> can be any of the following:

```
h      jump on A high
nle    jump on not low or equal
l      jump on A low
nhe    jump on not high or equal
ne     jump on A not equal B
lh     jump on low or high
e      jump on A equal B
nlh    jump on not low or high
nl     jump on A not low
he     jump on high or equal
nh     jump on A not high
le     jump on low or equal
```



File: as.info, Node: s390 Operand Modifier, Next: s390 Instruction Marker, Prev: s390 Aliases, Up: s390 Syntax

9.39.3.6 Instruction Operand Modifier

.....

If a symbol modifier is attached to a symbol in an expression for an instruction operand field, the symbol term is replaced with a reference to an object in the global offset table (GOT) or the procedure linkage table (PLT). The following expressions are allowed: 'symbol@modifier + constant', 'symbol@modifier + label + constant', and 'symbol@modifier - label + constant'. The term 'symbol' is the symbol that will be entered into the GOT or PLT, 'label' is a local label, and 'constant' is an arbitrary expression that the assembler can evaluate to a constant value.

The term '(symbol + constant1)@modifier +/- label + constant2' is also accepted but a warning message is printed and the term is converted to 'symbol@modifier +/- label + constant1 + constant2'.

'@got'
'@got12'

The @got modifier can be used for displacement fields, 16-bit immediate fields and 32-bit pc-relative immediate fields. The @got12 modifier is synonym to @got. The symbol is added to the GOT. For displacement fields and 16-bit immediate fields the symbol term is replaced with the offset from the start of the GOT to the GOT slot for the symbol. For a 32-bit pc-relative field the pc-relative offset to the GOT slot from the current instruction address is used.

'@gotent'

The @gotent modifier can be used for 32-bit pc-relative immediate fields. The symbol is added to the GOT and the symbol term is replaced with the pc-relative offset from the current instruction to the GOT slot for the symbol.

'@gotoff'

The @gotoff modifier can be used for 16-bit immediate fields. The symbol term is replaced with the offset from the start of the GOT to the address of the symbol.

'@gotplt'

The @gotplt modifier can be used for displacement fields, 16-bit immediate fields, and 32-bit pc-relative immediate fields. A procedure linkage table entry is generated for the symbol and a jump slot for the symbol is added to the GOT. For displacement fields and 16-bit immediate fields the symbol term is replaced with the offset from the start of the GOT to the jump slot for the symbol. For a 32-bit pc-relative field the pc-relative offset to the jump slot from the current instruction address is used.

'@plt'
The @plt modifier can be used for 16-bit and 32-bit pc-relative immediate fields. A procedure linkage table entry is generated for the symbol. The symbol term is replaced with the relative offset from the current instruction to the PLT entry for the symbol.

'@pltloff'
The @pltloff modifier can be used for 16-bit immediate fields. The symbol term is replaced with the offset from the start of the PLT to the address of the symbol.

'@gotntpoff'
The @gotntpoff modifier can be used for displacement fields. The symbol is added to the static TLS block and the negated offset to the symbol in the static TLS block is added to the GOT. The symbol term is replaced with the offset to the GOT slot from the start of the GOT.

'@indntpoff'
The @indntpoff modifier can be used for 32-bit pc-relative immediate fields. The symbol is added to the static TLS block and the negated offset to the symbol in the static TLS block is added to the GOT. The symbol term is replaced with the pc-relative offset to the GOT slot from the current instruction address.

For more information about the thread local storage modifiers 'gotntpoff' and 'indntpoff' see the ELF extension documentation 'ELF Handling For Thread-Local Storage'.



File: as.info, Node: s390 Instruction Marker, Next: s390 Literal Pool Entries, Prev: s390 Operand Modifier, Up: s390 Syntax

9.39.3.7 Instruction Marker

.....

The thread local storage instruction markers are used by the linker to perform code optimization.

':tls_load'
The :tls_load marker is used to flag the load instruction in the initial exec TLS model that retrieves the offset from the thread pointer to a thread local storage variable from the GOT.

':tls_gdcall'
The :tls_gdcall marker is used to flag the branch-and-save instruction to the __tls_get_offset function in the global dynamic TLS model.

':tls_ldcall'
The :tls_ldcall marker is used to flag the branch-and-save instruction to the __tls_get_offset function in the local dynamic TLS model.

For more information about the thread local storage instruction marker and the linker optimizations see the ELF extension documentation

'ELF Handling For Thread-Local Storage'.



File: as.info, Node: s390 Literal Pool Entries, Prev: s390 Instruction Marker, Up: s390 Syntax

9.39.3.8 Literal Pool Entries

.....

A literal pool is a collection of values. To access the values a pointer to the literal pool is loaded to a register, the literal pool register. Usually, register %r13 is used as the literal pool register (*note s390 Register:). Literal pool entries are created by adding the suffix :lit1, :lit2, :lit4, or :lit8 to the end of an expression for an instruction operand. The expression is added to the literal pool and the operand is replaced with the offset to the literal in the literal pool.

':lit1'

The literal pool entry is created as an 8-bit value. An operand modifier must not be used for the original expression.

':lit2'

The literal pool entry is created as a 16 bit value. The operand modifier @got may be used in the original expression. The term 'x@got:lit2' will put the got offset for the global symbol x to the literal pool as 16 bit value.

':lit4'

The literal pool entry is created as a 32-bit value. The operand modifier @got and @plt may be used in the original expression. The term 'x@got:lit4' will put the got offset for the global symbol x to the literal pool as a 32-bit value. The term 'x@plt:lit4' will put the plt offset for the global symbol x to the literal pool as a 32-bit value.

':lit8'

The literal pool entry is created as a 64-bit value. The operand modifier @got and @plt may be used in the original expression. The term 'x@got:lit8' will put the got offset for the global symbol x to the literal pool as a 64-bit value. The term 'x@plt:lit8' will put the plt offset for the global symbol x to the literal pool as a 64-bit value.

The assembler directive '.ltorg' is used to emit all literal pool entries to the current position.



File: as.info, Node: s390 Directives, Next: s390 Floating Point, Prev: s390 Syntax, Up: S/390-Dependent

9.39.4 Assembler Directives

'as' for s390 supports all of the standard ELF assembler directives as outlined in the main part of this document. Some directives have been extended and there are some additional directives, which are only available for the s390 'as'.

'.insn'

This directive permits the numeric representation of an instructions and makes the assembler insert the operands according

to one of the instructions formats for '.insn' (*note s390 Formats:). For example, the instruction 'l %r1,24(%r15)' could be written as '.insn rx,0x58000000,%r1,24(%r15)'.

'.short'

'.long'

'.quad'

This directive places one or more 16-bit (.short), 32-bit (.long), or 64-bit (.quad) values into the current section. If an ELF or TLS modifier is used only the following expressions are allowed: 'symbol@modifier + constant', 'symbol@modifier + label + constant', and 'symbol@modifier - label + constant'. The following modifiers are available:

'@got'

'@got12'

The @got modifier can be used for .short, .long and .quad.

The @got12 modifier is synonym to @got. The symbol is added to the GOT. The symbol term is replaced with offset from the start of the GOT to the GOT slot for the symbol.

'@gotoff'

The @gotoff modifier can be used for .short, .long and .quad.

The symbol term is replaced with the offset from the start of the GOT to the address of the symbol.

'@gotplt'

The @gotplt modifier can be used for .long and .quad. A procedure linkage table entry is generated for the symbol and a jump slot for the symbol is added to the GOT. The symbol term is replaced with the offset from the start of the GOT to the jump slot for the symbol.

'@plt'

The @plt modifier can be used for .long and .quad. A

procedure linkage table entry is generated for the symbol.

The symbol term is replaced with the address of the PLT entry for the symbol.

'@pltoff'

The @pltoff modifier can be used for .short, .long and .quad.

The symbol term is replaced with the offset from the start of the PLT to the address of the symbol.

'@tsgd'

'@tsgldm'

The @tsgd and @tsgldm modifier can be used for .long and .quad. A tls_index structure for the symbol is added to the GOT. The symbol term is replaced with the offset from the start of the GOT to the tls_index structure.

'@gotntpoff'

'@indntpoff'

The @gotntpoff and @indntpoff modifier can be used for .long and .quad. The symbol is added to the static TLS block and the negated offset to the symbol in the static TLS block is added to the GOT. For @gotntpoff the symbol term is replaced with the offset from the start of the GOT to the GOT slot, for @indntpoff the symbol term is replaced with the address of the GOT slot.

'@dtpoff'

The @dtpoff modifier can be used for .long and .quad. The symbol term is replaced with the offset of the symbol relative to the start of the TLS block it is contained in.

'@ntpoff'

The @ntpoff modifier can be used for .long and .quad. The symbol term is replaced with the offset of the symbol relative

to the TCB pointer.

For more information about the thread local storage modifiers see the ELF extension documentation 'ELF Handling For Thread-Local Storage'.

`!.ltorg'`

This directive causes the current contents of the literal pool to be dumped to the current location (*note s390 Literal Pool Entries:).

`!.machine STRING[+EXTENSION]...'`

This directive allows changing the machine for which code is generated. 'string' may be any of the '-march=' selection options, or 'push', or 'pop'. '.machine push' saves the currently selected cpu, which may be restored with '.machine pop'. Be aware that the cpu string has to be put into double quotes in case it contains characters not appropriate for identifiers. So you have to write '"z9-109"' instead of just 'z9-109'. Extensions can be specified after the cpu name, separated by plus characters. Valid extensions are: 'htm', 'nohtm', 'vx', 'novx'. They extend the basic instruction set with features from a higher cpu level, or remove support for a feature from the given cpu level.

Example: 'z13+nohtm' allows all instructions of the z13 cpu except instructions from the HTM facility.

`!.machinemode string'`

This directive allows to change the architecture mode for which code is being generated. 'string' may be 'esa', 'zarch', 'zarch_nohighgprs', 'push', or 'pop'. '.machinemode zarch_nohighgprs' can be used to prevent the 'highgprs' flag from being set in the ELF header of the output file. This is useful in situations where the code is gated with a runtime check which makes sure that the code is only executed on kernels providing the 'highgprs' feature. '.machinemode push' saves the currently selected mode, which may be restored with '.machinemode pop'.



File: as.info, Node: s390 Floating Point, Prev: s390 Directives, Up: S/390-Dependent

9.39.5 Floating Point

The assembler recognizes both the IEEE floating-point instruction and the hexadecimal floating-point instructions. The floating-point constructors '.float', '.single', and '.double' always emit the IEEE format. To assemble hexadecimal floating-point constants the '.long' and '.quad' directives must be used.



File: as.info, Node: SCORE-Dependent, Next: SH-Dependent, Prev: S/390-Dependent, Up: Machine Dependencies

9.40 SCORE Dependent Features

=====

* Menu:

* SCORE-Opts:: Assembler options
* SCORE-Pseudo:: SCORE Assembler Directives
* SCORE-Syntax:: Syntax



File: as.info, Node: SCORE-Opts, Next: SCORE-Pseudo, Up: SCORE-Dependent

9.40.1 Options

The following table lists all available SCORE options.

'-G NUM'

This option sets the largest size of an object that can be referenced implicitly with the 'gp' register. The default value is 8.

'-EB'

Assemble code for a big-endian cpu

'-EL'

Assemble code for a little-endian cpu

'-FIXDD'

Assemble code for fix data dependency

'-NWARN'

Assemble code for no warning message for fix data dependency

'-SCORE5'

Assemble code for target is SCORE5

'-SCORE5U'

Assemble code for target is SCORE5U

'-SCORE7'

Assemble code for target is SCORE7, this is default setting

'-SCORE3'

Assemble code for target is SCORE3

'-march=score7'

Assemble code for target is SCORE7, this is default setting

'-march=score3'

Assemble code for target is SCORE3

'-USE_R1'

Assemble code for no warning message when using temp register r1

'-KPIC'

Generate code for PIC. This option tells the assembler to generate score position-independent macro expansions. It also tells the assembler to mark the output file as PIC.

'-O0'

Assembler will not perform any optimizations

'-V'
Sunplus release version

US

File: as.info, Node: SCORE-Pseudo, Next: SCORE-Syntax, Prev: SCORE-Opts, Up: SCORE-Dependent

9.40.2 SCORE Assembler Directives

A number of assembler directives are available for SCORE. The following table is far from complete.

'**.set nwarn**'
Let the assembler not to generate warnings if the source machine language instructions happen data dependency.

'**.set fixdd**'
Let the assembler to insert bubbles (32 bit nop instruction / 16 bit nop! Instruction) if the source machine language instructions happen data dependency.

'**.set nofixdd**'
Let the assembler to generate warnings if the source machine language instructions happen data dependency. (Default)

'**.set r1**'
Let the assembler not to generate warnings if the source program uses r1. allow user to use r1

'**set nor1**'
Let the assembler to generate warnings if the source program uses r1. (Default)

'**.sdata**'
Tell the assembler to add subsequent data into the sdata section

'**.rdata**'
Tell the assembler to add subsequent data into the rdata section

'**.frame "frame-register", "offset", "return-pc-register"**'
Describe a stack frame. "frame-register" is the frame register, "offset" is the distance from the frame register to the virtual frame pointer, "return-pc-register" is the return program register. You must use ".ent" before ".frame" and only one ".frame" can be used per ".ent".

'**.mask "bitmask", "frameoffset"**'
Indicate which of the integer registers are saved in the current function's stack frame, this is for the debugger to explain the frame chain.

'**.ent "proc-name"**'
Set the beginning of the procedure "proc_name". Use this directive when you want to generate information for the debugger.

'**.end proc-name**'
Set the end of a procedure. Use this directive to generate


```
'--big'
    Generate big endian code.

'--relax'
    Alter jump instructions for long displacements.

'--small'
    Align sections to 4 byte boundaries, not 16.

'--dsp'
    Enable sh-dsp insns, and disable sh3e / sh4 insns.

'--renesas'
    Disable optimization with section symbol for compatibility with
    Renesas assembler.

'--allow-reg-prefix'
    Allow '$' as a register name prefix.

'--fdpic'
    Generate an FDPIC object file.

'--isa=sh4 | sh4a'
    Specify the sh4 or sh4a instruction set.

'--isa=dsp'
    Enable sh-dsp insns, and disable sh3e / sh4 insns.

'--isa=fp'
    Enable sh2e, sh3e, sh4, and sh4a insn sets.

'--isa=all'
    Enable sh1, sh2, sh2e, sh3, sh3e, sh4, sh4a, and sh-dsp insn sets.

'-h-tick-hex'
    Support H'00 style hex constants in addition to 0x00 style.
```



File: as.info, Node: SH Syntax, Next: SH Floating Point, Prev: SH Options, Up: SH-Dependent

9.41.2 Syntax

* Menu:

```
* SH-Chars::          Special Characters
* SH-Regs::           Register Names
* SH-Addressing::    Addressing Modes
```



File: as.info, Node: SH-Chars, Next: SH-Regs, Up: SH Syntax

9.41.2.1 Special Characters

.....

'!' is the line comment character.

You can use ';' instead of a newline to separate statements.

If a '#' appears as the first character of a line then the whole line

is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

Since '\$' has no special meaning, you may use it in symbol names.



File: as.info, Node: SH-Regs, Next: SH-Addressing, Prev: SH-Chars, Up: SH Syntax

9.41.2.2 Register Names

.....

You can use the predefined symbols 'r0', 'r1', 'r2', 'r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10', 'r11', 'r12', 'r13', 'r14', and 'r15' to refer to the SH registers.

The SH also has these control registers:

'pr'
procedure register (holds return address)

'pc'
program counter

'mach'
'macl'
high and low multiply accumulator registers

'sr'
status register

'gbr'
global base register

'vbr'
vector base register (for interrupt vectors)



File: as.info, Node: SH-Addressing, Prev: SH-Regs, Up: SH Syntax

9.41.2.3 Addressing Modes

.....

'as' understands the following addressing modes for the SH. 'RN' in the following refers to any of the numbered registers, but not the control registers.

'RN'
Register direct

'@RN'
Register indirect

'@-RN'
Register indirect with pre-decrement

'@RN+'
Register indirect with post-increment

'@(DISP, RN)'
Register indirect with displacement

'@(R0, RN)'
Register indexed

'@(DISP, GBR)'
'GBR' offset

'@(R0, GBR)'
GBR indexed

'ADDR'
'@(DISP, PC)'
PC relative address (for branch or for addressing memory). The 'as' implementation allows you to use the simpler form ADDR anywhere a PC relative address is called for; the alternate form is supported for compatibility with other assemblers.

'#IMM'
Immediate data



File: as.info, Node: SH Floating Point, Next: SH Directives, Prev: SH Syntax, Up: SH-Dependent

9.41.3 Floating Point

SH2E, SH3E and SH4 groups have on-chip floating-point unit (FPU). Other SH groups can use '.float' directive to generate IEEE floating-point numbers.

SH2E and SH3E support single-precision floating point calculations as well as entirely PCAPI compatible emulation of double-precision floating point calculations. SH2E and SH3E instructions are a subset of the floating point calculations conforming to the IEEE754 standard.

In addition to single-precision and double-precision floating-point operation capability, the on-chip FPU of SH4 has a 128-bit graphic engine that enables 32-bit floating-point data to be processed 128 bits at a time. It also supports 4 * 4 array operations and inner product operations. Also, a superscalar architecture is employed that enables simultaneous execution of two instructions (including FPU instructions), providing performance of up to twice that of conventional architectures at the same frequency.



File: as.info, Node: SH Directives, Next: SH Opcodes, Prev: SH Floating Point, Up: SH-Dependent

9.41.4 SH Machine Directives

'uaword'
'ualong'
'uaquad'

'as' will issue a warning when a misaligned '.word', '.long', or '.quad' directive is used. You may use '.uaword', '.ualong', or

'`.uquad`' to indicate that the value is intentionally misaligned.



File: as.info, Node: SH Opcodes, Prev: SH Directives, Up: SH-Dependent

9.41.5 Opcodes

For detailed information on the SH machine instruction set, see 'SH-Microcomputer User's Manual' (Renesas) or 'SH-4 32-bit CPU Core Architecture' (SuperH) and 'SuperH (SH) 64-Bit RISC Series' (SuperH).

'`as`' implements all the standard SH opcodes. No additional pseudo-instructions are needed on this family. Note, however, that because '`as`' supports a simpler form of PC-relative addressing, you may simply write (for example)

```
mov.l bar,r0
```

where other assemblers might require an explicit displacement to '`bar`' from the program counter:

```
mov.l @(DISP, PC)
```

Here is a summary of SH opcodes:

Legend:

```
Rn      a numbered register
Rm      another numbered register
#imm    immediate data
disp    displacement
disp8   8-bit displacement
disp12  12-bit displacement
```

<code>add #imm,Rn</code>	<code>lds.l @Rn+,PR</code>
<code>add Rm,Rn</code>	<code>mac.w @Rm+,@Rn+</code>
<code>addc Rm,Rn</code>	<code>mov #imm,Rn</code>
<code>addv Rm,Rn</code>	<code>mov Rm,Rn</code>
<code>and #imm,R0</code>	<code>mov.b Rm,@(R0,Rn)</code>
<code>and Rm,Rn</code>	<code>mov.b Rm,@-Rn</code>
<code>and.b #imm,@(R0,GBR)</code>	<code>mov.b Rm,@Rn</code>
<code>bf disp8</code>	<code>mov.b @(disp,Rm),R0</code>
<code>bra disp12</code>	<code>mov.b @(disp,GBR),R0</code>
<code>bsr disp12</code>	<code>mov.b @(R0,Rm),Rn</code>
<code>bt disp8</code>	<code>mov.b @Rm+,Rn</code>
<code>clrmac</code>	<code>mov.b @Rm,Rn</code>
<code>clrt</code>	<code>mov.b R0,@(disp,Rm)</code>
<code>cmp/eq #imm,R0</code>	<code>mov.b R0,@(disp,GBR)</code>
<code>cmp/eq Rm,Rn</code>	<code>mov.l Rm,@(disp,Rn)</code>
<code>cmp/ge Rm,Rn</code>	<code>mov.l Rm,@(R0,Rn)</code>
<code>cmp/gt Rm,Rn</code>	<code>mov.l Rm,@-Rn</code>
<code>cmp/hi Rm,Rn</code>	<code>mov.l Rm,@Rn</code>
<code>cmp/hs Rm,Rn</code>	<code>mov.l @(disp,Rn),Rm</code>
<code>cmp/pl Rn</code>	<code>mov.l @(disp,GBR),R0</code>
<code>cmp/pz Rn</code>	<code>mov.l @(disp,PC),Rn</code>
<code>cmp/str Rm,Rn</code>	<code>mov.l @(R0,Rm),Rn</code>
<code>div0s Rm,Rn</code>	<code>mov.l @Rm+,Rn</code>
<code>div0u</code>	<code>mov.l @Rm,Rn</code>
<code>div1 Rm,Rn</code>	<code>mov.l R0,@(disp,GBR)</code>

exts.b Rm,Rn	mov.w Rm,@(R0,Rn)
exts.w Rm,Rn	mov.w Rm,@-Rn
extu.b Rm,Rn	mov.w Rm,@Rn
extu.w Rm,Rn	mov.w @(disp,Rm),R0
jmp @Rn	mov.w @(disp,GBR),R0
jsr @Rn	mov.w @(disp,PC),Rn
ldc Rn,GBR	mov.w @(R0,Rm),Rn
ldc Rn,SR	mov.w @Rm+,Rn
ldc Rn,VBR	mov.w @Rm,Rn
ldc.l @Rn+,GBR	mov.w R0,@(disp,Rm)
ldc.l @Rn+,SR	mov.w R0,@(disp,GBR)
ldc.l @Rn+,VBR	mova @(disp,PC),R0
lds Rn,MACH	movt Rn
lds Rn,MACL	muls Rm,Rn
lds Rn,PR	mulu Rm,Rn
lds.l @Rn+,MACH	neg Rm,Rn
lds.l @Rn+,MACL	negc Rm,Rn
nop	stc VBR,Rn
not Rm,Rn	stc.l GBR,@-Rn
or #imm,R0	stc.l SR,@-Rn
or Rm,Rn	stc.l VBR,@-Rn
or.b #imm,@(R0,GBR)	sts MACH,Rn
rotcl Rn	sts MACL,Rn
rotcr Rn	sts PR,Rn
rotl Rn	sts.l MACH,@-Rn
rotr Rn	sts.l MACL,@-Rn
rte	sts.l PR,@-Rn
rts	sub Rm,Rn
sett	subc Rm,Rn
shal Rn	subv Rm,Rn
shar Rn	swap.b Rm,Rn
shll Rn	swap.w Rm,Rn
shll16 Rn	tas.b @Rn
shll2 Rn	trapa #imm
shll8 Rn	tst #imm,R0
shlr Rn	tst Rm,Rn
shlr16 Rn	tst.b #imm,@(R0,GBR)
shlr2 Rn	xor #imm,R0
shlr8 Rn	xor Rm,Rn
sleep	xor.b #imm,@(R0,GBR)
stc GBR,Rn	xtrct Rm,Rn
stc SR,Rn	



File: as.info, Node: SH64-Dependent, Next: Sparc-Dependent, Prev: SH-Dependent, Up: Machine Dependencies

9.42 SuperH SH64 Dependent Features

=====

* Menu:

* SH64 Options::	Options
* SH64 Syntax::	Syntax
* SH64 Directives::	SH64 Machine Directives
* SH64 Opcodes::	Opcodes



File: as.info, Node: SH64 Options, Next: SH64 Syntax, Up: SH64-Dependent

9.42.1 Options

'-isa=sh4 | sh4a'
Specify the sh4 or sh4a instruction set.

'-isa=dsp'
Enable sh-dsp insns, and disable sh3e / sh4 insns.

'-isa=fp'
Enable sh2e, sh3e, sh4, and sh4a insn sets.

'-isa=all'
Enable sh1, sh2, sh2e, sh3, sh3e, sh4, sh4a, and sh-dsp insn sets.

'-isa=shmedia | -isa=shcompact'
Specify the default instruction set. 'SHmedia' specifies the 32-bit opcodes, and 'SHcompact' specifies the 16-bit opcodes compatible with previous SH families. The default depends on the ABI selected; the default for the 64-bit ABI is SHmedia, and the default for the 32-bit ABI is SHcompact. If neither the ABI nor the ISA is specified, the default is 32-bit SHcompact.

Note that the '.mode' pseudo-op is not permitted if the ISA is not specified on the command line.

'-abi=32 | -abi=64'
Specify the default ABI. If the ISA is specified and the ABI is not, the default ABI depends on the ISA, with SHmedia defaulting to 64-bit and SHcompact defaulting to 32-bit.

Note that the '.abi' pseudo-op is not permitted if the ABI is not specified on the command line. When the ABI is specified on the command line, any '.abi' pseudo-ops in the source must match it.

'-shcompact-const-crange'
Emit code-range descriptors for constants in SHcompact code sections.

'-no-mix'
Disallow SHmedia code in the same section as constants and SHcompact code.

'-no-expand'
Do not expand MOVI, PT, PTA or PTB instructions.

'-expand-pt32'
With -abi=64, expand PT, PTA and PTB instructions to 32 bits only.

'-h-tick-hex'
Support H'00 style hex constants in addition to 0x00 style.



File: as.info, Node: SH64 Syntax, Next: SH64 Directives, Prev: SH64 Options, Up: SH64-Dependent

9.42.2 Syntax

* Menu:

* SH64-Chars:: Special Characters

* SH64-Regs:: Register Names
 * SH64-Addressing:: Addressing Modes



File: as.info, Node: SH64-Chars, Next: SH64-Regs, Up: SH64 Syntax

9.42.2.1 Special Characters

.....

'!' is the line comment character.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

You can use ';' instead of a newline to separate statements.

Since '\$' has no special meaning, you may use it in symbol names.



File: as.info, Node: SH64-Regs, Next: SH64-Addressing, Prev: SH64-Chars, Up: SH64 Syntax

9.42.2.2 Register Names

.....

You can use the predefined symbols 'r0' through 'r63' to refer to the SH64 general registers, 'cr0' through 'cr63' for control registers, 'tr0' through 'tr7' for target address registers, 'fr0' through 'fr63' for single-precision floating point registers, 'dr0' through 'dr62' (even numbered registers only) for double-precision floating point registers, 'fv0' through 'fv60' (multiples of four only) for single-precision floating point vectors, 'fp0' through 'fp62' (even numbered registers only) for single-precision floating point pairs, 'mtrx0' through 'mtrx48' (multiples of 16 only) for 4x4 matrices of single-precision floating point registers, 'pc' for the program counter, and 'fpscr' for the floating point status and control register.

You can also refer to the control registers by the mnemonics 'sr', 'ssr', 'pssr', 'intevt', 'expevt', 'pexpevt', 'tra', 'spc', 'pspc', 'resvec', 'vbr', 'tea', 'dcr', 'kcr0', 'kcr1', 'ctc', and 'usr'.



File: as.info, Node: SH64-Addressing, Prev: SH64-Regs, Up: SH64 Syntax

9.42.2.3 Addressing Modes

.....

SH64 operands consist of either a register or immediate value. The immediate value can be a constant or label reference (or portion of a label reference), as in this example:

```
movi    4,r2
pt      function, tr4
movi    (function >> 16) & 65535,r0
shori   function & 65535, r0
ld.l    r0,4,r0
```


Instruction label references can reference labels in either SHmedia or SHcompact. To differentiate between the two, labels in SHmedia sections will always have the least significant bit set (i.e. they will be odd), which SHcompact labels will have the least significant bit reset (i.e. they will be even). If you need to reference the actual address of a label, you can use the 'datalabel' modifier, as in this example:

```
.long    function
.long    datalabel function
```

In that example, the first longword may or may not have the least significant bit set depending on whether the label is an SHmedia label or an SHcompact label. The second longword will be the actual address of the label, regardless of what type of label it is.

US

File: as.info, Node: SH64 Directives, Next: SH64 Opcodes, Prev: SH64 Syntax, Up: SH64-Dependent

9.42.3 SH64 Machine Directives

In addition to the SH directives, the SH64 provides the following directives:

```
' .mode [shmedia|shcompact]'
```

```
' .isa [shmedia|shcompact]'
```

Specify the ISA for the following instructions (the two directives are equivalent). Note that programs such as 'objdump' rely on symbolic labels to determine when such mode switches occur (by checking the least significant bit of the label's address), so such mode/isa changes should always be followed by a label (in practice, this is true anyway). Note that you cannot use these directives if you didn't specify an ISA on the command line.

```
' .abi [32|64]'
```

Specify the ABI for the following instructions. Note that you cannot use this directive unless you specified an ABI on the command line, and the ABIs specified must match.

US

File: as.info, Node: SH64 Opcodes, Prev: SH64 Directives, Up: SH64-Dependent

9.42.4 Opcodes

For detailed information on the SH64 machine instruction set, see 'SuperH 64 bit RISC Series Architecture Manual' (SuperH, Inc.).

'as' implements all the standard SH64 opcodes. In addition, the following pseudo-opcodes may be expanded into one or more alternate opcodes:

```
'movi'
```

If the value doesn't fit into a standard 'movi' opcode, 'as' will replace the 'movi' with a sequence of 'movi' and 'shori' opcodes.

```
'pt'
```

This expands to a sequence of 'movi' and 'shori' opcode, followed by a 'ptrel' opcode, or to a 'pta' or 'ptb' opcode, depending on the label referenced.

US

File: as.info, Node: Sparc-Dependent, Next: TIC54X-Dependent, Prev: SH64-Dependent, Up: Machine Dependencies

9.43 SPARC Dependent Features

=====

* Menu:

* Sparc-Opts:: Options
 * Sparc-Aligned-Data:: Option to enforce aligned data
 * Sparc-Syntax:: Syntax
 * Sparc-Float:: Floating Point
 * Sparc-Directives:: Sparc Machine Directives

US

File: as.info, Node: Sparc-Opts, Next: Sparc-Aligned-Data, Up: Sparc-Dependent

9.43.1 Options

The SPARC chip family includes several successive versions, using the same core instruction set, but including a few additional instructions at each version. There are exceptions to this however. For details on what instructions each variant supports, please see the chip's architecture reference manual.

By default, 'as' assumes the core instruction set (SPARC v6), but "bumps" the architecture level as needed: it switches to successively higher architectures as it encounters instructions that only exist in the higher levels.

If not configured for SPARC v9 ('sparc64-*-*') GAS will not bump past sparclite by default, an option must be passed to enable the v9 instructions.

GAS treats sparclite as being compatible with v8, unless an architecture is explicitly requested. SPARC v9 is always incompatible with sparclite.

```
'-Av6 | -Av7 | -Av8 | -Aleon | -Asparclet | -Asparclite'  

'-Av8plus | -Av8plusa | -Av8plusb | -Av8plusc | -Av8plused | -Av8plusv | -Av8plusm'  

'-Av9 | -Av9a | -Av9b | -Av9c | -Av9d | -Av9e | -Av9v | -Av9m'  

'-Asparc | -Asparcvis | -Asparcvis2 | -Asparcfmaf | -Asparcima'  

'-Asparcvis3 | -Asparcvis3r | -Asparc5'
```

Use one of the '-A' options to select one of the SPARC architectures explicitly. If you select an architecture explicitly, 'as' reports a fatal error if it encounters an instruction or feature requiring an incompatible or higher level.

'-Av8plus', '-Av8plusa', '-Av8plusb', '-Av8plusc', '-Av8plused', and '-Av8plusv' select a 32 bit environment.

'-Av9', '-Av9a', '-Av9b', '-Av9c', '-Av9d', '-Av9e', '-Av9v' and '-Av9m' select a 64 bit environment and are not available unless

GAS is explicitly configured with 64 bit environment support.

'-Av8plusa' and '-Av9a' enable the SPARC V9 instruction set with UltraSPARC VIS 1.0 extensions.

'-Av8plusb' and '-Av9b' enable the UltraSPARC VIS 2.0 instructions, as well as the instructions enabled by '-Av8plusa' and '-Av9a'.

'-Av8plusc' and '-Av9c' enable the UltraSPARC Niagara instructions, as well as the instructions enabled by '-Av8plusb' and '-Av9b'.

'-Av8plUSD' and '-Av9d' enable the floating point fused multiply-add, VIS 3.0, and HPC extension instructions, as well as the instructions enabled by '-Av8plusc' and '-Av9c'.

'-Av8pluse' and '-Av9e' enable the cryptographic instructions, as well as the instructions enabled by '-Av8plUSD' and '-Av9d'.

'-Av8plusv' and '-Av9v' enable floating point unfused multiply-add, and integer multiply-add, as well as the instructions enabled by '-Av8pluse' and '-Av9e'.

'-Av8plusm' and '-Av9m' enable the VIS 4.0, subtract extended, xmpmul, xmntmul and xmntsqr instructions, as well as the instructions enabled by '-Av8plusv' and '-Av9v'.

'-Asparc' specifies a v9 environment. It is equivalent to '-Av9' if the word size is 64-bit, and '-Av8plus' otherwise.

'-Asparcvis' specifies a v9a environment. It is equivalent to '-Av9a' if the word size is 64-bit, and '-Av8plusa' otherwise.

'-Asparcvis2' specifies a v9b environment. It is equivalent to '-Av9b' if the word size is 64-bit, and '-Av8plusb' otherwise.

'-Asparcfmaf' specifies a v9b environment with the floating point fused multiply-add instructions enabled.

'-Asparcima' specifies a v9b environment with the integer multiply-add instructions enabled.

'-Asparcvis3' specifies a v9b environment with the VIS 3.0, HPC, and floating point fused multiply-add instructions enabled.

'-Asparcvis3r' specifies a v9b environment with the VIS 3.0, HPC, and floating point unfused multiply-add instructions enabled.

'-Asparc5' is equivalent to '-Av9m'.

```
'-xarch=v8plus | -xarch=v8plusa | -xarch=v8plusb | -xarch=v8plusc'
'-xarch=v8plUSD | -xarch=v8plusv | -xarch=v8plusm | -xarch=v9 | -xarch=v9a'
'-xarch=v9b | -xarch=v9c | -xarch=v9d | -xarch=v9e | -xarch=v9v | -xarch=v9m'
'-xarch=sparc | -xarch=sparcvis | -xarch=sparcvis2'
'-xarch=sparcfmaf | -xarch=sparcima | -xarch=sparcvis3'
'-xarch=sparcvis3r | -xarch=sparc5'
```

For compatibility with the SunOS v9 assembler. These options are equivalent to -Av8plus, -Av8plusa, -Av8plusb, -Av8plusc, -Av8plUSD, -Av8plusv, -Av8plusm, -Av9, -Av9a, -Av9b, -Av9c, -Av9d, -Av9e, -Av9v, -Av9m, -Asparc, -Asparcvis, -Asparcvis2, -Asparcfmaf,

-Asparcima, -Asparcvis3, and -Asparcvis3r, respectively.

'-bump'

Warn whenever it is necessary to switch to another level. If an architecture level is explicitly requested, GAS will not issue warnings until that level is reached, and will then bump the level as required (except between incompatible levels).

'-32 | -64'

Select the word size, either 32 bits or 64 bits. These options are only available with the ELF object file format, and require that the necessary BFD support has been included.

'--dcti-couples-detect'

Warn if a DCTI (delayed control transfer instruction) couple is found when generating code for a variant of the SPARC architecture in which the execution of the couple is unpredictable, or very slow. This is disabled by default.



File: as.info, Node: Sparc-Aligned-Data, Next: Sparc-Syntax, Prev: Sparc-Opts, Up: Sparc-Dependent

9.43.2 Enforcing aligned data

SPARC GAS normally permits data to be misaligned. For example, it permits the '.long' pseudo-op to be used on a byte boundary. However, the native SunOS assemblers issue an error when they see misaligned data.

You can use the '--enforce-aligned-data' option to make SPARC GAS also issue an error about misaligned data, just as the SunOS assemblers do.

The '--enforce-aligned-data' option is not the default because gcc issues misaligned data pseudo-ops when it initializes certain packed data structures (structures defined using the 'packed' attribute). You may have to assemble with GAS in order to initialize packed data structures in your own code.



File: as.info, Node: Sparc-Syntax, Next: Sparc-Float, Prev: Sparc-Aligned-Data, Up: Sparc-Dependent

9.43.3 Sparc Syntax

The assembler syntax closely follows The Sparc Architecture Manual, versions 8 and 9, as well as most extensions defined by Sun for their UltraSPARC and Niagara line of processors.

* Menu:

* Sparc-Chars::	Special Characters
* Sparc-Regs::	Register Names
* Sparc-Constants::	Constant Names
* Sparc-Relocs::	Relocations
* Sparc-Size-Translations::	Size Translations



File: as.info, Node: Sparc-Chars, Next: Sparc-Regs, Up: Sparc-Syntax

9.43.3.1 Special Characters

.....

A '!' character appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

';' can be used instead of a newline to separate statements.



File: as.info, Node: Sparc-Regs, Next: Sparc-Constants, Prev: Sparc-Chars, Up: Sparc-Syntax

9.43.3.2 Register Names

.....

The Sparc integer register file is broken down into global, outgoing, local, and incoming.

- * The 8 global registers are referred to as '%gN'.
- * The 8 outgoing registers are referred to as '%oN'.
- * The 8 local registers are referred to as '%lN'.
- * The 8 incoming registers are referred to as '%iN'.
- * The frame pointer register '%i6' can be referenced using the alias '%fp'.
- * The stack pointer register '%o6' can be referenced using the alias '%sp'.

Floating point registers are simply referred to as '%fN'. When assembling for pre-V9, only 32 floating point registers are available. For V9 and later there are 64, but there are restrictions when referencing the upper 32 registers. They can only be accessed as double or quad, and thus only even or quad numbered accesses are allowed. For example, '%f34' is a legal floating point register, but '%f35' is not.

Floating point registers accessed as double can also be referred using the '%dN' notation, where N is even. Similarly, floating point registers accessed as quad can be referred using the '%qN' notation, where N is a multiple of 4. For example, '%f4' can be denoted as both '%d4' and '%q4'. On the other hand, '%f2' can be denoted as '%d2' but not as '%q2'.

Certain V9 instructions allow access to ancillary state registers. Most simply they can be referred to as '%asrN' where N can be from 16 to 31. However, there are some aliases defined to reference ASR registers defined for various UltraSPARC processors:

- * The tick compare register is referred to as '%tick_cmpr'.
- * The system tick register is referred to as '%stick'. An alias, '%sys_tick', exists but is deprecated and should not be used by new software.
- * The system tick compare register is referred to as '%stick_cmpr'. An alias, '%sys_tick_cmpr', exists but is deprecated and should not be used by new software.
- * The software interrupt register is referred to as '%softint'.
- * The set software interrupt register is referred to as '%set_softint'. The mnemonic '%softint_set' is provided as an alias.
- * The clear software interrupt register is referred to as '%clear_softint'. The mnemonic '%softint_clear' is provided as an alias.
- * The performance instrumentation counters register is referred to as '%pic'.
- * The performance control register is referred to as '%pcr'.
- * The graphics status register is referred to as '%gsr'.
- * The V9 dispatch control register is referred to as '%dcr'.

Various V9 branch and conditional move instructions allow specification of which set of integer condition codes to test. These are referred to as '%xcc' and '%icc'.

Additionally, GAS supports the so-called "natural" condition codes; these are referred to as '%ncc' and reference to '%icc' if the word size is 32, '%xcc' if the word size is 64.

In V9, there are 4 sets of floating point condition codes which are referred to as '%fccN'.

Several special privileged and non-privileged registers exist:

- * The V9 address space identifier register is referred to as '%asi'.
- * The V9 restorable windows register is referred to as '%canrestore'.
- * The V9 savable windows register is referred to as '%cansave'.
- * The V9 clean windows register is referred to as '%cleanwin'.
- * The V9 current window pointer register is referred to as '%cwp'.
- * The floating-point queue register is referred to as '%fq'.
- * The V8 co-processor queue register is referred to as '%cq'.
- * The floating point status register is referred to as '%fsr'.

- * The other windows register is referred to as '%otherwin'.
- * The V9 program counter register is referred to as '%pc'.
- * The V9 next program counter register is referred to as '%npc'.
- * The V9 processor interrupt level register is referred to as '%pil'.
- * The V9 processor state register is referred to as '%pstate'.
- * The trap base address register is referred to as '%tba'.
- * The V9 tick register is referred to as '%tick'.
- * The V9 trap level is referred to as '%tl'.
- * The V9 trap program counter is referred to as '%tpc'.
- * The V9 trap next program counter is referred to as '%tnpc'.
- * The V9 trap state is referred to as '%tstate'.
- * The V9 trap type is referred to as '%tt'.
- * The V9 condition codes is referred to as '%ccr'.
- * The V9 floating-point registers state is referred to as '%fprs'.
- * The V9 version register is referred to as '%ver'.
- * The V9 window state register is referred to as '%wstate'.
- * The Y register is referred to as '%y'.
- * The V8 window invalid mask register is referred to as '%wim'.
- * The V8 processor state register is referred to as '%psr'.
- * The V9 global register level register is referred to as '%gl'.

Several special register names exist for hypervisor mode code:

- * The hyperprivileged processor state register is referred to as '%hpstate'.
- * The hyperprivileged trap state register is referred to as '%htstate'.
- * The hyperprivileged interrupt pending register is referred to as '%hintp'.
- * The hyperprivileged trap base address register is referred to as '%htba'.
- * The hyperprivileged implementation version register is referred to as '%hver'.
- * The hyperprivileged system tick offset register is referred to as '%hstick_offset'. Note that there is no '%hstick' register, the

normal '%stick' is used.

- * The hyperprivileged system tick enable register is referred to as '%hstick_enable'.
- * The hyperprivileged system tick compare register is referred to as '%hstick_cmp'.



File: as.info, Node: Sparc-Constants, Next: Sparc-Relocs, Prev: Sparc-Regs, Up: Sparc-Syntax

9.43.3.3 Constants

.....

Several Sparc instructions take an immediate operand field for which mnemonic names exist. Two such examples are 'membar' and 'prefetch'. Another example are the set of V9 memory access instruction that allow specification of an address space identifier.

The 'membar' instruction specifies a memory barrier that is the defined by the operand which is a bitmask. The supported mask mnemonics are:

- * '#Sync' requests that all operations (including nonmemory reference operations) appearing prior to the 'membar' must have been performed and the effects of any exceptions become visible before any instructions after the 'membar' may be initiated. This corresponds to 'membar' cmask field bit 2.
- * '#MemIssue' requests that all memory reference operations appearing prior to the 'membar' must have been performed before any memory operation after the 'membar' may be initiated. This corresponds to 'membar' cmask field bit 1.
- * '#Lookaside' requests that a store appearing prior to the 'membar' must complete before any load following the 'membar' referencing the same address can be initiated. This corresponds to 'membar' cmask field bit 0.
- * '#StoreStore' defines that the effects of all stores appearing prior to the 'membar' instruction must be visible to all processors before the effect of any stores following the 'membar'. Equivalent to the deprecated 'stbar' instruction. This corresponds to 'membar' mmask field bit 3.
- * '#LoadStore' defines all loads appearing prior to the 'membar' instruction must have been performed before the effect of any stores following the 'membar' is visible to any other processor. This corresponds to 'membar' mmask field bit 2.
- * '#StoreLoad' defines that the effects of all stores appearing prior to the 'membar' instruction must be visible to all processors before loads following the 'membar' may be performed. This corresponds to 'membar' mmask field bit 1.
- * '#LoadLoad' defines that all loads appearing prior to the 'membar' instruction must have been performed before any loads following the 'membar' may be performed. This corresponds to 'membar' mmask

field bit 0.

These values can be ored together, for example:

```
membar #Sync
membar #StoreLoad | #LoadLoad
membar #StoreLoad | #StoreStore
```

The 'prefetch' and 'prefetcha' instructions take a prefetch function code. The following prefetch function code constant mnemonics are available:

- * '#n_reads' requests a prefetch for several reads, and corresponds to a prefetch function code of 0.
- '#one_read' requests a prefetch for one read, and corresponds to a prefetch function code of 1.
- '#n_writes' requests a prefetch for several writes (and possibly reads), and corresponds to a prefetch function code of 2.
- '#one_write' requests a prefetch for one write, and corresponds to a prefetch function code of 3.
- '#page' requests a prefetch page, and corresponds to a prefetch function code of 4.
- '#invalidate' requests a prefetch invalidate, and corresponds to a prefetch function code of 16.
- '#unified' requests a prefetch to the nearest unified cache, and corresponds to a prefetch function code of 17.
- '#n_reads_strong' requests a strong prefetch for several reads, and corresponds to a prefetch function code of 20.
- '#one_read_strong' requests a strong prefetch for one read, and corresponds to a prefetch function code of 21.
- '#n_writes_strong' requests a strong prefetch for several writes, and corresponds to a prefetch function code of 22.
- '#one_write_strong' requests a strong prefetch for one write, and corresponds to a prefetch function code of 23.

Only one prefetch code may be specified. Here are some examples:

```
prefetch [%l0 + %l2], #one_read
prefetch [%g2 + 8], #n_writes
prefetcha [%g1] 0x8, #unified
prefetcha [%o0 + 0x10] %asi, #n_reads
```

The actual behavior of a given prefetch function code is processor specific. If a processor does not implement a given prefetch function code, it will treat the prefetch instruction as a nop.

For instructions that accept an immediate address space identifier, 'as' provides many mnemonics corresponding to V9 defined as well as UltraSPARC and Niagara extended values. For example, '#ASI_P' and

'#ASI_BLK_INIT_QUAD_LDD_AIUS'. See the V9 and processor specific manuals for details.



File: as.info, Node: Sparc-Relocs, Next: Sparc-Size-Translations, Prev: Sparc-Constants, Up: Sparc-Syntax

9.43.3.4 Relocations

.....

ELF relocations are available as defined in the 32-bit and 64-bit Sparc ELF specifications.

'R_SPARC_HI22' is obtained using '%hi' and 'R_SPARC_LO10' is obtained using '%lo'. Likewise 'R_SPARC_HIX22' is obtained from '%hix' and 'R_SPARC_LOX10' is obtained using '%lox'. For example:

```
sethi %hi(symbol), %g1
or    %g1, %lo(symbol), %g1

sethi %hix(symbol), %g1
xor   %g1, %lox(symbol), %g1
```

These "high" mnemonics extract bits 31:10 of their operand, and the "low" mnemonics extract bits 9:0 of their operand.

V9 code model relocations can be requested as follows:

- * 'R_SPARC_HH22' is requested using '%hh'. It can also be generated using '%uhi'.
- * 'R_SPARC_HM10' is requested using '%hm'. It can also be generated using '%ulo'.
- * 'R_SPARC_LM22' is requested using '%lm'.
- * 'R_SPARC_H44' is requested using '%h44'.
- * 'R_SPARC_M44' is requested using '%m44'.
- * 'R_SPARC_L44' is requested using '%l44' or '%l34'.
- * 'R_SPARC_H34' is requested using '%h34'.

The '%l34' generates a 'R_SPARC_L44' relocation because it calculates the necessary value, and therefore no explicit 'R_SPARC_L34' relocation needed to be created for this purpose.

The '%h34' and '%l34' relocations are used for the abs34 code model. Here is an example abs34 address generation sequence:

```
sethi %h34(symbol), %g1
sllx %g1, 2, %g1
or    %g1, %l34(symbol), %g1
```

The PC relative relocation 'R_SPARC_PC22' can be obtained by enclosing an operand inside of '%pc22'. Likewise, the 'R_SPARC_PC10' relocation can be obtained using '%pc10'. These are mostly used when assembling PIC code. For example, the standard PIC sequence on Sparc to get the base of the global offset table, PC relative, into a register, can be performed as:

```
sethi %pc22(_GLOBAL_OFFSET_TABLE_-4), %l7
add   %l7, %pc10(_GLOBAL_OFFSET_TABLE_+4), %l7
```

Several relocations exist to allow the link editor to potentially optimize GOT data references. The 'R_SPARC_GOTDATA_OP_HIX22' relocation can be obtained by enclosing an operand inside of '%gdop_hix22'. The 'R_SPARC_GOTDATA_OP_LOX10' relocation can be obtained by enclosing an operand inside of '%gdop_lox10'. Likewise, 'R_SPARC_GOTDATA_OP' can be obtained by enclosing an operand inside of '%gdop'. For example, assuming the GOT base is in register '%l7':

```
sethi %gdop_hix22(symbol), %l1
xor   %l1, %gdop_lox10(symbol), %l1
ld    [%l7 + %l1], %l2, %gdop(symbol)
```

There are many relocations that can be requested for access to thread local storage variables. All of the Sparc TLS mnemonics are supported:

```
* 'R_SPARC_TLS_GD_HI22' is requested using '%tgd_hi22'.
* 'R_SPARC_TLS_GD_LO10' is requested using '%tgd_lo10'.
* 'R_SPARC_TLS_GD_ADD' is requested using '%tgd_add'.
* 'R_SPARC_TLS_GD_CALL' is requested using '%tgd_call'.

* 'R_SPARC_TLS_LDM_HI22' is requested using '%tldm_hi22'.
* 'R_SPARC_TLS_LDM_LO10' is requested using '%tldm_lo10'.
* 'R_SPARC_TLS_LDM_ADD' is requested using '%tldm_add'.
* 'R_SPARC_TLS_LDM_CALL' is requested using '%tldm_call'.

* 'R_SPARC_TLS_LDO_HIX22' is requested using '%tlldo_hix22'.
* 'R_SPARC_TLS_LDO_LOX10' is requested using '%tlldo_lox10'.
* 'R_SPARC_TLS_LDO_ADD' is requested using '%tlldo_add'.

* 'R_SPARC_TLS_IE_HI22' is requested using '%tie_hi22'.
* 'R_SPARC_TLS_IE_LO10' is requested using '%tie_lo10'.
* 'R_SPARC_TLS_IE_LD' is requested using '%tie_ld'.
* 'R_SPARC_TLS_IE_LDX' is requested using '%tie_ldx'.
* 'R_SPARC_TLS_IE_ADD' is requested using '%tie_add'.

* 'R_SPARC_TLS_LE_HIX22' is requested using '%tle_hix22'.
* 'R_SPARC_TLS_LE_LOX10' is requested using '%tle_lox10'.
```

Here are some example TLS model sequences.

First, General Dynamic:

```
sethi %tgd_hi22(symbol), %l1
add   %l1, %tgd_lo10(symbol), %l1
add   %l7, %l1, %o0, %tgd_add(symbol)
call  __tls_get_addr, %tgd_call(symbol)
nop
```

Local Dynamic:

```
sethi %tldm_hi22(symbol), %l1
add   %l1, %tldm_lo10(symbol), %l1
add   %l7, %l1, %o0, %tldm_add(symbol)
call  __tls_get_addr, %tldm_call(symbol)
nop

sethi %tlldo_hix22(symbol), %l1
xor   %l1, %tlldo_lox10(symbol), %l1
```

```
add    %o0, %l1, %l1, %tldo_add(symbol)
```

Initial Exec:

```
sethi  %tie_hi22(symbol), %l1
add    %l1, %tie_lo10(symbol), %l1
ld     [%l7 + %l1], %o0, %tie_ld(symbol)
add    %g7, %o0, %o0, %tie_add(symbol)
```

```
sethi  %tie_hi22(symbol), %l1
add    %l1, %tie_lo10(symbol), %l1
ldx   [%l7 + %l1], %o0, %tie_ldx(symbol)
add    %g7, %o0, %o0, %tie_add(symbol)
```

And finally, Local Exec:

```
sethi  %tle_hix22(symbol), %l1
add    %l1, %tle_lox10(symbol), %l1
add    %g7, %l1, %l1
```

When assembling for 64-bit, and a secondary constant addend is specified in an address expression that would normally generate an 'R_SPARC_L010' relocation, the assembler will emit an 'R_SPARC_OL010' instead.



File: as.info, Node: Sparc-Size-Translations, Prev: Sparc-Relocs, Up: Sparc-Syntax

9.43.3.5 Size Translations

.....

Often it is desirable to write code in an operand size agnostic manner. 'as' provides support for this via operand size opcode translations. Translations are supported for loads, stores, shifts, compare-and-swap atomics, and the 'clr' synthetic instruction.

If generating 32-bit code, 'as' will generate the 32-bit opcode. Whereas if 64-bit code is being generated, the 64-bit opcode will be emitted. For example 'ldn' will be transformed into 'ld' for 32-bit code and 'ldx' for 64-bit code.

Here is an example meant to demonstrate all the supported opcode translations:

```
ldn   [%o0], %o1
ldna  [%o0] %asi, %o2
stn   %o1, [%o0]
stna  %o2, [%o0] %asi
slln  %o3, 3, %o3
srln  %o4, 8, %o4
sran  %o5, 12, %o5
casn  [%o0], %o1, %o2
casna [%o0] %asi, %o1, %o2
clrn  %g1
```

In 32-bit mode 'as' will emit:

```
ld    [%o0], %o1
lda  [%o0] %asi, %o2
```

```

st   %o1, [%o0]
sta  %o2, [%o0] %asi
sll  %o3, 3, %o3
srl  %o4, 8, %o4
sra  %o5, 12, %o5
cas  [%o0], %o1, %o2
casa [%o0] %asi, %o1, %o2
clr  %g1

```

And in 64-bit mode 'as' will emit:

```

ldx  [%o0], %o1
ldxa [%o0] %asi, %o2
stx  %o1, [%o0]
stxa %o2, [%o0] %asi
sllx %o3, 3, %o3
srlx %o4, 8, %o4
srax %o5, 12, %o5
casx [%o0], %o1, %o2
casxa [%o0] %asi, %o1, %o2
clrx %g1

```

Finally, the '.nword' translating directive is supported as well. It is documented in the section on Sparc machine directives.



File: as.info, Node: Sparc-Float, Next: Sparc-Directives, Prev: Sparc-Syntax, Up: Sparc-Dependent

9.43.4 Floating Point

The Sparc uses IEEE floating-point numbers.



File: as.info, Node: Sparc-Directives, Prev: Sparc-Float, Up: Sparc-Dependent

9.43.5 Sparc Machine Directives

The Sparc version of 'as' supports the following additional machine directives:

'`.align`'

This must be followed by the desired alignment in bytes.

'`.common`'

This must be followed by a symbol name, a positive number, and "`bss`". This behaves somewhat like '`.comm`', but the syntax is different.

'`.half`'

This is functionally identical to '`.short`'.

'`.nword`'

On the Sparc, the '`.nword`' directive produces native word sized value, ie. if assembling with -32 it is equivalent to '`.word`', if assembling with -64 it is equivalent to '`.xword`'.

' .proc'

This directive is ignored. Any text following it on the same line is also ignored.

' .register'

This directive declares use of a global application or system register. It must be followed by a register name %g2, %g3, %g6 or %g7, comma and the symbol name for that register. If symbol name is '#scratch', it is a scratch register, if it is '#ignore', it just suppresses any errors about using undeclared global register, but does not emit any information about it into the object file. This can be useful e.g. if you save the register before use and restore it after.

' .reserve'

This must be followed by a symbol name, a positive number, and '"bss"'. This behaves somewhat like '.lcomm', but the syntax is different.

' .seg'

This must be followed by '"text"', '"data"', or '"data1"'. It behaves like '.text', '.data', or '.data 1'.

' .skip'

This is functionally identical to the '.space' directive.

' .word'

On the Sparc, the '.word' directive produces 32 bit values, instead of the 16 bit values it produces on many other machines.

' .xword'

On the Sparc V9 processor, the '.xword' directive produces 64 bit values.



File: as.info, Node: TIC54X-Dependent, Next: TIC6X-Dependent, Prev: Sparc-Dependent, Up: Machine Dependencies

9.44 TIC54X Dependent Features

=====

* Menu:

* TIC54X-Opts::	Command-line Options
* TIC54X-Block::	Blocking
* TIC54X-Env::	Environment Settings
* TIC54X-Constants::	Constants Syntax
* TIC54X-Subsyms::	String Substitution
* TIC54X-Locals::	Local Label Syntax
* TIC54X-Builtins::	Builtin Assembler Math Functions
* TIC54X-Ext::	Extended Addressing Support
* TIC54X-Directives::	Directives
* TIC54X-Macros::	Macro Features
* TIC54X-MMMRegs::	Memory-mapped Registers
* TIC54X-Syntax::	Syntax



File: as.info, Node: TIC54X-Opts, Next: TIC54X-Block, Up: TIC54X-Dependent

9.44.1 Options

The TMS320C54X version of 'as' has a few machine-dependent options.

You can use the '-mfar-mode' option to enable extended addressing mode. All addresses will be assumed to be > 16 bits, and the appropriate relocation types will be used. This option is equivalent to using the '.far_mode' directive in the assembly code. If you do not use the '-mfar-mode' option, all references will be assumed to be 16 bits. This option may be abbreviated to '-mf'.

You can use the '-mcpu' option to specify a particular CPU. This option is equivalent to using the '.version' directive in the assembly code. For recognized CPU codes, see *Note '.version': TIC54X-Directives. The default CPU version is '542'.

You can use the '-merrors-to-file' option to redirect error output to a file (this provided for those deficient environments which don't provide adequate output redirection). This option may be abbreviated to '-me'.



File: as.info, Node: TIC54X-Block, Next: TIC54X-Env, Prev: TIC54X-Opts, Up: TIC54X-Dependent

9.44.2 Blocking

A blocked section or memory block is guaranteed not to cross the blocking boundary (usually a page, or 128 words) if it is smaller than the blocking size, or to start on a page boundary if it is larger than the blocking size.



File: as.info, Node: TIC54X-Env, Next: TIC54X-Constants, Prev: TIC54X-Block, Up: TIC54X-Dependent

9.44.3 Environment Settings

'C54XDSP_DIR' and 'A_DIR' are semicolon-separated paths which are added to the list of directories normally searched for source and include files. 'C54XDSP_DIR' will override 'A_DIR'.



File: as.info, Node: TIC54X-Constants, Next: TIC54X-Subsyms, Prev: TIC54X-Env, Up: TIC54X-Dependent

9.44.4 Constants Syntax

The TIC54X version of 'as' allows the following additional constant formats, using a suffix to indicate the radix:

Binary	000000B, 011000b
Octal	10Q, 224q
Hexadecimal	45h, 0FH



File: as.info, Node: TIC54X-Subsyms, Next: TIC54X-Locals, Prev: TIC54X-Constants,
Up: TIC54X-Dependent

9.44.5 String Substitution

A subset of allowable symbols (which we'll call subsyms) may be assigned arbitrary string values. This is roughly equivalent to C preprocessor #define macros. When 'as' encounters one of these symbols, the symbol is replaced in the input stream by its string value. Subsym names *must* begin with a letter.

Subsyms may be defined using the '.asg' and '.eval' directives (*Note '.asg': TIC54X-Directives, *Note '.eval': TIC54X-Directives.

Expansion is recursive until a previously encountered symbol is seen, at which point substitution stops.

In this example, x is replaced with SYM2; SYM2 is replaced with SYM1, and SYM1 is replaced with x. At this point, x has already been encountered and the substitution stops.

```
.asg  "x",SYM1
.asg  "SYM1",SYM2
.asg  "SYM2",x
add   x,a           ; final code assembled is "add x, a"
```

Macro parameters are converted to subsyms; a side effect of this is the normal 'as' '\ARG' dereferencing syntax is unnecessary. Subsyms defined within a macro will have global scope, unless the '.var' directive is used to identify the subsym as a local macro variable (*note '.var': TIC54X-Directives.

Substitution may be forced in situations where replacement might be ambiguous by placing colons on either side of the subsym. The following code:

```
.eval  "10",x
LAB:X: add   #x, a
```

When assembled becomes:

```
LAB10 add   #10, a
```

Smaller parts of the string assigned to a subsym may be accessed with the following syntax:

':SYMBOL(CHAR_INDEX):'

Evaluates to a single-character string, the character at CHAR_INDEX.

':SYMBOL(START,LENGTH):'

Evaluates to a substring of SYMBOL beginning at START with length LENGTH.



File: as.info, Node: TIC54X-Locals, Next: TIC54X-Builtins, Prev: TIC54X-Subsyms,
Up: TIC54X-Dependent

9.44.6 Local Labels

Local labels may be defined in two ways:

- * \$N, where N is a decimal number between 0 and 9
- * LABEL?, where LABEL is any legal symbol name.

Local labels thus defined may be redefined or automatically generated. The scope of a local label is based on when it may be undefined or reset. This happens when one of the following situations is encountered:

- * .newblock directive *note '.newblock': TIC54X-Directives.
- * The current section is changed (.sect, .text, or .data)
- * Entering or leaving an included file
- * The macro scope where the label was defined is exited



File: as.info, Node: TIC54X-Builtins, Next: TIC54X-Ext, Prev: TIC54X-Locals, Up: TIC54X-Dependent

9.44.7 Math Builtins

The following built-in functions may be used to generate a floating-point value. All return a floating-point value except '\$cvi', '\$int', and '\$sgn', which return an integer value.

- '\$acos(EXPR)'
Returns the floating point arccosine of EXPR.
- '\$asin(EXPR)'
Returns the floating point arcsine of EXPR.
- '\$atan(EXPR)'
Returns the floating point arctangent of EXPR.
- '\$atan2(EXPR1,EXPR2)'
Returns the floating point arctangent of EXPR1 / EXPR2.
- '\$ceil(EXPR)'
Returns the smallest integer not less than EXPR as floating point.
- '\$cosh(EXPR)'
Returns the floating point hyperbolic cosine of EXPR.
- '\$cos(EXPR)'
Returns the floating point cosine of EXPR.
- '\$cvf(EXPR)'
Returns the integer value EXPR converted to floating-point.
- '\$cvi(EXPR)'
Returns the floating point value EXPR converted to integer.
- '\$exp(EXPR)'
Returns the floating point value $e ^ EXPR$.

'\$fabs(EXPR)'
Returns the floating point absolute value of EXPR.

'\$floor(EXPR)'
Returns the largest integer that is not greater than EXPR as floating point.

'\$fmod(EXPR1,EXPR2)'
Returns the floating point remainder of EXPR1 / EXPR2.

'\$int(EXPR)'
Returns 1 if EXPR evaluates to an integer, zero otherwise.

'\$ldexp(EXPR1,EXPR2)'
Returns the floating point value $EXPR1 * 2 ^ EXPR2$.

'\$log10(EXPR)'
Returns the base 10 logarithm of EXPR.

'\$log(EXPR)'
Returns the natural logarithm of EXPR.

'\$max(EXPR1,EXPR2)'
Returns the floating point maximum of EXPR1 and EXPR2.

'\$min(EXPR1,EXPR2)'
Returns the floating point minimum of EXPR1 and EXPR2.

'\$pow(EXPR1,EXPR2)'
Returns the floating point value $EXPR1 ^ EXPR2$.

'\$round(EXPR)'
Returns the nearest integer to EXPR as a floating point number.

'\$sgn(EXPR)'
Returns -1, 0, or 1 based on the sign of EXPR.

'\$sin(EXPR)'
Returns the floating point sine of EXPR.

'\$sinh(EXPR)'
Returns the floating point hyperbolic sine of EXPR.

'\$sqrt(EXPR)'
Returns the floating point square root of EXPR.

'\$tan(EXPR)'
Returns the floating point tangent of EXPR.

'\$tanh(EXPR)'
Returns the floating point hyperbolic tangent of EXPR.

'\$trunc(EXPR)'
Returns the integer value of EXPR truncated towards zero as floating point.

File: as.info, Node: TIC54X-Ext, Next: TIC54X-Directives, Prev: TIC54X-Builtins,
Up: TIC54X-Dependent

9.44.8 Extended Addressing

The 'LDX' pseudo-op is provided for loading the extended addressing bits of a label or address. For example, if an address '_label' resides in extended program memory, the value of '_label' may be loaded as follows:

```
ldx    #_label,16,a    ; loads extended bits of _label
or     #_label,a      ; loads lower 16 bits of _label
bacc   a              ; full address is in accumulator A
```



File: as.info, Node: TIC54X-Directives, Next: TIC54X-Macros, Prev: TIC54X-Ext,
Up: TIC54X-Dependent

9.44.9 Directives

'**.align [SIZE]**'

'**.even**'

Align the section program counter on the next boundary, based on SIZE. SIZE may be any power of 2. '**.even**' is equivalent to '**.align**' with a SIZE of 2.

'1'

Align SPC to word boundary

'2'

Align SPC to longword boundary (same as **.even**)

'128'

Align SPC to page boundary

'**.asg STRING, NAME**'

Assign NAME the string STRING. String replacement is performed on STRING before assignment.

'**.eval STRING, NAME**'

Evaluate the contents of string STRING and assign the result as a string to the subsym NAME. String replacement is performed on STRING before assignment.

'**.bss SYMBOL, SIZE [, [BLOCKING_FLAG] [,ALIGNMENT_FLAG]]**'

Reserve space for SYMBOL in the **.bss** section. SIZE is in words. If present, **BLOCKING_FLAG** indicates the allocated space should be aligned on a page boundary if it would otherwise cross a page boundary. If present, **ALIGNMENT_FLAG** causes the assembler to allocate SIZE on a long word boundary.

'**.byte VALUE [,...,VALUE_N]**'

'**.ubyte VALUE [,...,VALUE_N]**'

'**.char VALUE [,...,VALUE_N]**'

'**.uchar VALUE [,...,VALUE_N]**'

Place one or more bytes into consecutive words of the current section. The upper 8 bits of each word is zero-filled. If a label is used, it points to the word allocated for the first byte encountered.

'**.clink ["SECTION_NAME"]**'

Set **STYP_CLINK** flag for this section, which indicates to the linker

that if no symbols from this section are referenced, the section should not be included in the link. If SECTION_NAME is omitted, the current section is used.

`'.c_mode'`
TBD.

`'.copy "FILENAME" | FILENAME'`

`'.include "FILENAME" | FILENAME'`

Read source statements from FILENAME. The normal include search path is used. Normally `.copy` will cause statements from the included file to be printed in the assembly listing and `.include` will not, but this distinction is not currently implemented.

`'.data'`

Begin assembling code into the `.data` section.

`'.double VALUE [, ..., VALUE_N]'`

`'.ldouble VALUE [, ..., VALUE_N]'`

`'.float VALUE [, ..., VALUE_N]'`

`'.xfloat VALUE [, ..., VALUE_N]'`

Place an IEEE single-precision floating-point representation of one or more floating-point values into the current section. All but `'.xfloat'` align the result on a longword boundary. Values are stored most-significant word first.

`'.drlist'`

`'.drnolist'`

Control printing of directives to the listing file. Ignored.

`'.emsg STRING'`

`'.mmsg STRING'`

`'.wmsg STRING'`

Emit a user-defined error, message, or warning, respectively.

`'.far_mode'`

Use extended addressing when assembling statements. This should appear only once per file, and is equivalent to the `-mfar-mode` option `*note '-mfar-mode': TIC54X-Opts`.

`'.fclist'`

`'.fcnolist'`

Control printing of false conditional blocks to the listing file.

`'.field VALUE [, SIZE]'`

Initialize a bitfield of SIZE bits in the current section. If VALUE is relocatable, then SIZE must be 16. SIZE defaults to 16 bits. If VALUE does not fit into SIZE bits, the value will be truncated. Successive `'.field'` directives will pack starting at the current word, filling the most significant bits first, and aligning to the start of the next word if the field size does not fit into the space remaining in the current word. A `'.align'` directive with an operand of 1 will force the next `'.field'` directive to begin packing into a new word. If a label is used, it points to the word that contains the specified field.

`'.global SYMBOL [, ..., SYMBOL_N]'`

`'.def SYMBOL [, ..., SYMBOL_N]'`

`'.ref SYMBOL [, ..., SYMBOL_N]'`

'`.def`' nominally identifies a symbol defined in the current file and available to other files. '`.ref`' identifies a symbol used in the current file but defined elsewhere. Both map to the standard '`.global`' directive.

'`.half VALUE [, ..., VALUE_N]`'
'`.uhalf VALUE [, ..., VALUE_N]`'
'`.short VALUE [, ..., VALUE_N]`'
'`.ushort VALUE [, ..., VALUE_N]`'
'`.int VALUE [, ..., VALUE_N]`'
'`.uint VALUE [, ..., VALUE_N]`'
'`.word VALUE [, ..., VALUE_N]`'
'`.uword VALUE [, ..., VALUE_N]`'

Place one or more values into consecutive words of the current section. If a label is used, it points to the word allocated for the first value encountered.

'`.label SYMBOL`'

Define a special SYMBOL to refer to the load time address of the current section program counter.

'`.length`'

'`.width`'

Set the page length and width of the output listing file. Ignored.

'`.list`'

'`.nolist`'

Control whether the source listing is printed. Ignored.

'`.long VALUE [, ..., VALUE_N]`'

'`.ulong VALUE [, ..., VALUE_N]`'

'`.xlong VALUE [, ..., VALUE_N]`'

Place one or more 32-bit values into consecutive words in the current section. The most significant word is stored first.

'`.long`' and '`.ulong`' align the result on a longword boundary;

'`.xlong`' does not.

'`.loop [COUNT]`'

'`.break [CONDITION]`'

'`.endloop`'

Repeatedly assemble a block of code. '`.loop`' begins the block, and '`.endloop`' marks its termination. COUNT defaults to 1024, and indicates the number of times the block should be repeated.

'`.break`' terminates the loop so that assembly begins after the '`.endloop`' directive. The optional CONDITION will cause the loop to terminate only if it evaluates to zero.

'`MACRO_NAME .macro [PARAM1][, ..., PARAM_N]`'

'`[.mexit]`'

'`.endm`'

See the section on macros for more explanation (*Note TIC54X-Macros::).

'`.mlib "FILENAME" | FILENAME`'

Load the macro library FILENAME. FILENAME must be an archived library (BFD ar-compatible) of text files, expected to contain only macro definitions. The standard include search path is used.

'`.mlist`'

`'.mnolist'`
Control whether to include macro and loop block expansions in the listing output. Ignored.

`'.mmregs'`
Define global symbolic names for the 'c54x registers. Supposedly equivalent to executing `'.set'` directives for each register with its memory-mapped value, but in reality is provided only for compatibility and does nothing.

`'.newblock'`
This directive resets any TIC54X local labels currently defined. Normal 'as' local labels are unaffected.

`'.option OPTION_LIST'`
Set listing options. Ignored.

`'.sblock "SECTION_NAME" | SECTION_NAME [,"NAME_N" | NAME_N]'`
Designate SECTION_NAME for blocking. Blocking guarantees that a section will start on a page boundary (128 words) if it would otherwise cross a page boundary. Only initialized sections may be designated with this directive. See also `*Note TIC54X-Block::`.

`'.sect "SECTION_NAME"'`
Define a named initialized section and make it the current section.

`'SYMBOL .set "VALUE"'`
`'SYMBOL .equ "VALUE"'`
Equate a constant VALUE to a SYMBOL, which is placed in the symbol table. SYMBOL may not be previously defined.

`'.space SIZE_IN_BITS'`
`'.bes SIZE_IN_BITS'`
Reserve the given number of bits in the current section and zero-fill them. If a label is used with `'.space'`, it points to the `*first*` word reserved. With `'.bes'`, the label points to the `*last*` word reserved.

`'.sslist'`
`'.ssnolist'`
Controls the inclusion of subsym replacement in the listing output. Ignored.

`'.string "STRING" [,...,"STRING_N]'`
`'.pstring "STRING" [,...,"STRING_N]'`
Place 8-bit characters from STRING into the current section. `'.string'` zero-fills the upper 8 bits of each word, while `'.pstring'` puts two characters into each word, filling the most-significant bits first. Unused space is zero-filled. If a label is used, it points to the first word initialized.

`'[STAG] .struct [OFFSET]'`
`'[NAME_1] element [COUNT_1]'`
`'[NAME_2] element [COUNT_2]'`
`'[TNAME] .tag STAGX [TCOUNT]'`
`'...'`
`'[NAME_N] element [COUNT_N]'`
`'[SSIZE] .endstruct'`
`'LABEL .tag [STAG]'`

Assign symbolic offsets to the elements of a structure. STAG defines a symbol to use to reference the structure. OFFSET indicates a starting value to use for the first element encountered; otherwise it defaults to zero. Each element can have a named offset, NAME, which is a symbol assigned the value of the element's offset into the structure. If STAG is missing, these become global symbols. COUNT adjusts the offset that many times, as if 'element' were an array. 'element' may be one of '.byte', '.word', '.long', '.float', or any equivalent of those, and the structure offset is adjusted accordingly. '.field' and '.string' are also allowed; the size of '.field' is one bit, and '.string' is considered to be one word in size. Only element descriptors, structure/union tags, '.align' and conditional assembly directives are allowed within '.struct'/'endstruct'. '.align' aligns member offsets to word boundaries only. SSIZE, if provided, will always be assigned the size of the structure.

The '.tag' directive, in addition to being used to define a structure/union element within a structure, may be used to apply a structure to a symbol. Once applied to LABEL, the individual structure elements may be applied to LABEL to produce the desired offsets using LABEL as the structure base.

'.tab'

Set the tab size in the output listing. Ignored.

```
'[UTAG] .union'
'[NAME_1] element [COUNT_1]'
'[NAME_2] element [COUNT_2]'
'[TNAME] .tag UTAGX[,TCOUNT]'
'...'
'[NAME_N] element [COUNT_N]'
'[USIZE] .endstruct'
'LABEL .tag [UTAG]'
```

Similar to '.struct', but the offset after each element is reset to zero, and the USIZE is set to the maximum of all defined elements. Starting offset for the union is always zero.

```
'[SYMBOL] .usect "SECTION_NAME", SIZE, [, [BLOCKING_FLAG] [, ALIGNMENT_FLAG]]'
```

Reserve space for variables in a named, uninitialized section (similar to .bss). '.usect' allows definitions sections independent of .bss. SYMBOL points to the first location reserved by this allocation. The symbol may be used as a variable name. SIZE is the allocated size in words. BLOCKING_FLAG indicates whether to block this section on a page boundary (128 words) (*note TIC54X-Block:). ALIGNMENT FLAG indicates whether the section should be longword-aligned.

```
'.var SYM[,..., SYM_N]'
```

Define a subsym to be a local variable within a macro. See *Note TIC54X-Macros:.

```
'.version VERSION'
```

Set which processor to build instructions for. Though the following values are accepted, the op is ignored.

```
'541'
'542'
'543'
'545'
```

```
'545LP'  
'546LP'  
'548'  
'549'
```



File: as.info, Node: TIC54X-Macros, Next: TIC54X-MMRregs, Prev: TIC54X-Directives,
Up: TIC54X-Dependent

9.44.10 Macros

Macros do not require explicit dereferencing of arguments (i.e., \ARG).

During macro expansion, the macro parameters are converted to subsyms. If the number of arguments passed the macro invocation exceeds the number of parameters defined, the last parameter is assigned the string equivalent of all remaining arguments. If fewer arguments are given than parameters, the missing parameters are assigned empty strings. To include a comma in an argument, you must enclose the argument in quotes.

The following built-in subsym functions allow examination of the string value of subsyms (or ordinary strings). The arguments are strings unless otherwise indicated (subsyms passed as args will be replaced by the strings they represent).

'\$symlen(STR)'

Returns the length of STR.

'\$symcmp(STR1,STR2)'

Returns 0 if STR1 == STR2, non-zero otherwise.

'\$firstch(STR,CH)'

Returns index of the first occurrence of character constant CH in STR.

'\$lastch(STR,CH)'

Returns index of the last occurrence of character constant CH in STR.

'\$isdefed(SYMBOL)'

Returns zero if the symbol SYMBOL is not in the symbol table, non-zero otherwise.

'\$ismember(SYMBOL,LIST)'

Assign the first member of comma-separated string LIST to SYMBOL; LIST is reassigned the remainder of the list. Returns zero if LIST is a null string. Both arguments must be subsyms.

'\$iscons(EXPR)'

Returns 1 if string EXPR is binary, 2 if octal, 3 if hexadecimal, 4 if a character, 5 if decimal, and zero if not an integer.

'\$isname(NAME)'

Returns 1 if NAME is a valid symbol name, zero otherwise.

'\$isreg(REG)'

Returns 1 if REG is a valid predefined register name (AR0-AR7 only).

'\$structsz(STAG)'

Returns the size of the structure or union represented by STAG.

'\$structacc(STAG)'

Returns the reference point of the structure or union represented by STAG. Always returns zero.

US

File: as.info, Node: TIC54X-MMRegs, Next: TIC54X-Syntax, Prev: TIC54X-Macros, Up: TIC54X-Dependent

9.44.11 Memory-mapped Registers

The following symbols are recognized as memory-mapped registers:

US

File: as.info, Node: TIC54X-Syntax, Prev: TIC54X-MMRegs, Up: TIC54X-Dependent

9.44.12 TIC54X Syntax

* Menu:

* TIC54X-Chars:: Special Characters

US

File: as.info, Node: TIC54X-Chars, Up: TIC54X-Syntax

9.44.12.1 Special Characters

.....

The presence of a ';' appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The presence of an asterisk ('*') at the start of a line also indicates a comment that extends to the end of that line.

The TIC54X assembler does not currently support a line separator character.

US

File: as.info, Node: TIC6X-Dependent, Next: TILE-Gx-Dependent, Prev: TIC54X-Dependent, Up: Machine Dependencies

9.45 TIC6X Dependent Features

=====

* Menu:

* TIC6X Options:: Options
 * TIC6X Syntax:: Syntax
 * TIC6X Directives:: Directives



File: as.info, Node: TIC6X Options, Next: TIC6X Syntax, Up: TIC6X-Dependent

9.45.1 TIC6X Options

'-march=ARCH'

Enable (only) instructions from architecture ARCH. By default, all instructions are permitted.

The following values of ARCH are accepted: 'c62x', 'c64x', 'c64x+', 'c67x', 'c67x+', 'c674x'.

'-mdsbt'

'-mno-dsbt'

The '-mdsbt' option causes the assembler to generate the 'Tag_ABI_DSBT' attribute with a value of 1, indicating that the code is using DSBT addressing. The '-mno-dsbt' option, the default, causes the tag to have a value of 0, indicating that the code does not use DSBT addressing. The linker will emit a warning if objects of different type (DSBT and non-DSBT) are linked together.

'-mpid=no'

'-mpid=near'

'-mpid=far'

The '-mpid=' option causes the assembler to generate the 'Tag_ABI_PID' attribute with a value indicating the form of data addressing used by the code. '-mpid=no', the default, indicates position-dependent data addressing, '-mpid=near' indicates position-independent addressing with GOT accesses using near DP addressing, and '-mpid=far' indicates position-independent addressing with GOT accesses using far DP addressing. The linker will emit a warning if objects built with different settings of this option are linked together.

'-mpic'

'-mno-pic'

The '-mpic' option causes the assembler to generate the 'Tag_ABI_PIC' attribute with a value of 1, indicating that the code is using position-independent code addressing. The '-mno-pic' option, the default, causes the tag to have a value of 0, indicating position-dependent code addressing. The linker will emit a warning if objects of different type (position-dependent and position-independent) are linked together.

'-mbig-endian'

'-mlittle-endian'

Generate code for the specified endianness. The default is little-endian.



File: as.info, Node: TIC6X Syntax, Next: TIC6X Directives, Prev: TIC6X Options, Up: TIC6X-Dependent

9.45.2 TIC6X Syntax

The presence of a ';' on a line indicates the start of a comment that extends to the end of the current line. If a '#' or '*' appears as the first character of a line, the whole line is treated as a comment. Note that if a line starts with a '#' character then it can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The '@' character can be used instead of a newline to separate statements.

Instruction, register and functional unit names are case-insensitive. 'as' requires fully-specified functional unit names, such as '.S1', '.L1X' or '.D1T2', on all instructions using a functional unit.

For some instructions, there may be syntactic ambiguity between register or functional unit names and the names of labels or other symbols. To avoid this, enclose the ambiguous symbol name in parentheses; register and functional unit names may not be enclosed in parentheses.

US

File: as.info, Node: TIC6X Directives, Prev: TIC6X Syntax, Up: TIC6X-Dependent

9.45.3 TIC6X Directives

Directives controlling the set of instructions accepted by the assembler have effect for instructions between the directive and any subsequent directive overriding it.

'**.arch ARCH**'

This has the same effect as '-march=ARCH'.

'**.cantunwind**'

Prevents unwinding through the current function. No personality routine or exception table data is required or permitted.

If this is not specified then frame unwinding information will be constructed from CFI directives. *note CFI directives::.

'**.c6xabi_attribute TAG, VALUE**'

Set the C6000 EABI build attribute TAG to VALUE.

The TAG is either an attribute number or one of 'Tag_ISA', 'Tag_ABI_wchar_t', 'Tag_ABI_stack_align_needed', 'Tag_ABI_stack_align_preserved', 'Tag_ABI_DSBT', 'Tag_ABI_PID', 'Tag_ABI_PIC', 'TAG_ABI_array_object_alignment', 'TAG_ABI_array_object_align_expected', 'Tag_ABI_compatibility' and 'Tag_ABI_conformance'. The VALUE is either a 'number', '"string"', or 'number, "string"' depending on the tag.

'**.ehtype SYMBOL**'

Output an exception type table reference to SYMBOL.

'**.endp**'

Marks the end of and exception table or function. If preceded by a '.handlerdata' directive then this also switched back to the previous text section.

' .handlerdata'

Marks the end of the current function, and the start of the exception table entry for that function. Anything between this directive and the '.endp' directive will be added to the exception table entry.

Must be preceded by a CFI block containing a '.cfi_lsda' directive.

' .nocmp'

Disallow use of C64x+ compact instructions in the current text section.

' .personalityindex INDEX'

Sets the personality routine for the current function to the ABI specified compact routine number INDEX

' .personality NAME'

Sets the personality routine for the current function to NAME.

' .scomm SYMBOL, SIZE, ALIGN'

Like '.comm', creating a common symbol SYMBOL with size SIZE and alignment ALIGN, but unlike when using '.comm', this symbol will be placed into the small BSS section by the linker.



File: as.info, Node: TILE-Gx-Dependent, Next: TILEPro-Dependent, Prev: TIC6X-Dependent, Up: Machine Dependencies

9.46 TILE-Gx Dependent Features

=====

* Menu:

- * TILE-Gx Options:: TILE-Gx Options
- * TILE-Gx Syntax:: TILE-Gx Syntax
- * TILE-Gx Directives:: TILE-Gx Directives



File: as.info, Node: TILE-Gx Options, Next: TILE-Gx Syntax, Up: TILE-Gx-Dependent

9.46.1 Options

The following table lists all available TILE-Gx specific options:

' -m32 | -m64'

Select the word size, either 32 bits or 64 bits.

' -EB | -EL'

Select the endianness, either big-endian (-EB) or little-endian (-EL).



File: as.info, Node: TILE-Gx Syntax, Next: TILE-Gx Directives, Prev: TILE-Gx Options, Up: TILE-Gx-Dependent

9.46.2 Syntax

Block comments are delimited by `/*` and `*/`. End of line comments may be introduced by `#`.

Instructions consist of a leading opcode or macro name followed by whitespace and an optional comma-separated list of operands:

```
OPCODE [OPERAND, ...]
```

Instructions must be separated by a newline or semicolon.

There are two ways to write code: either write naked instructions, which the assembler is free to combine into VLIW bundles, or specify the VLIW bundles explicitly.

Bundles are specified using curly braces:

```
{ ADD r3,r4,r5 ; ADD r7,r8,r9 ; LW r10,r11 }
```

A bundle can span multiple lines. If you want to put multiple instructions on a line, whether in a bundle or not, you need to separate them with semicolons as in this example.

A bundle may contain one or more instructions, up to the limit specified by the ISA (currently three). If fewer instructions are specified than the hardware supports in a bundle, the assembler inserts `'fnop'` instructions automatically.

The assembler will prefer to preserve the ordering of instructions within the bundle, putting the first instruction in a lower-numbered pipeline than the next one, etc. This fact, combined with the optional use of explicit `'fnop'` or `'nop'` instructions, allows precise control over which pipeline executes each instruction.

If the instructions cannot be bundled in the listed order, the assembler will automatically try to find a valid pipeline assignment. If there is no way to bundle the instructions together, the assembler reports an error.

The assembler does not yet auto-bundle (automatically combine multiple instructions into one bundle), but it reserves the right to do so in the future. If you want to force an instruction to run by itself, put it in a bundle explicitly with curly braces and use `'nop'` instructions (not `'fnop'`) to fill the remaining pipeline slots in that bundle.

* Menu:

```
* TILE-Gx Opcodes::          Opcode Naming Conventions.
* TILE-Gx Registers::       Register Naming.
* TILE-Gx Modifiers::       Symbolic Operand Modifiers.
```



File: as.info, Node: TILE-Gx Opcodes, Next: TILE-Gx Registers, Up: TILE-Gx Syntax

9.46.2.1 Opcode Names

.....

For a complete list of opcodes and descriptions of their semantics, see `'TILE-Gx Instruction Set Architecture'`, available upon request at

www.tilera.com.



File: as.info, Node: TILE-Gx Registers, Next: TILE-Gx Modifiers, Prev: TILE-Gx Opcodes, Up: TILE-Gx Syntax

9.46.2.2 Register Names

.....

General-purpose registers are represented by predefined symbols of the form 'rN', where N represents a number between '0' and '63'. However, the following registers have canonical names that must be used instead:

```
'r54'
    sp
'r55'
    lr
'r56'
    sn
'r57'
    idn0
'r58'
    idn1
'r59'
    udn0
'r60'
    udn1
'r61'
    udn2
'r62'
    udn3
'r63'
    zero
```

The assembler will emit a warning if a numeric name is used instead of the non-numeric name. The `'.no_require_canonical_reg_names'` assembler pseudo-op turns off this warning. `'.require_canonical_reg_names'` turns it back on.



File: as.info, Node: TILE-Gx Modifiers, Prev: TILE-Gx Registers, Up: TILE-Gx Syntax

9.46.2.3 Symbolic Operand Modifiers

.....

The assembler supports several modifiers when using symbol addresses in TILE-Gx instruction operands. The general syntax is the following:

```
modifier(symbol)
```

The following modifiers are supported:

'hw0'

This modifier is used to load bits 0-15 of the symbol's address.

'hw1'

This modifier is used to load bits 16-31 of the symbol's address.

'hw2'

This modifier is used to load bits 32-47 of the symbol's address.

'hw3'

This modifier is used to load bits 48-63 of the symbol's address.

'hw0_last'

This modifier yields the same value as 'hw0', but it also checks that the value does not overflow.

'hw1_last'

This modifier yields the same value as 'hw1', but it also checks that the value does not overflow.

'hw2_last'

This modifier yields the same value as 'hw2', but it also checks that the value does not overflow.

A 48-bit symbolic value is constructed by using the following idiom:

```
moveli r0, hw2_last(sym)
shl16insli r0, r0, hw1(sym)
shl16insli r0, r0, hw0(sym)
```

'hw0_got'

This modifier is used to load bits 0-15 of the symbol's offset in the GOT entry corresponding to the symbol.

'hw0_last_got'

This modifier yields the same value as 'hw0_got', but it also checks that the value does not overflow.

'hw1_last_got'

This modifier is used to load bits 16-31 of the symbol's offset in the GOT entry corresponding to the symbol, and it also checks that the value does not overflow.

'plt'

This modifier is used for function symbols. It causes a `_procedure linkage table_`, an array of code stubs, to be created at the time the shared object is created or linked against, together with a global offset table entry. The value is a pc-relative offset to the corresponding stub code in the procedure linkage table. This arrangement causes the run-time symbol resolver to be called to look up and set the value of the symbol the first time the function is called (at latest; depending environment variables). It is only safe to leave the symbol unresolved this way if all references are function calls.

'hw0_plt'

This modifier is used to load bits 0-15 of the pc-relative address of a plt entry.

'hw1_plt'

This modifier is used to load bits 16-31 of the pc-relative address of a plt entry.

'hw1_last_plt'

This modifier yields the same value as 'hw1_plt', but it also checks that the value does not overflow.

'hw2_last_plt'

This modifier is used to load bits 32-47 of the pc-relative address of a plt entry, and it also checks that the value does not overflow.

'hw0_tls_gd'

This modifier is used to load bits 0-15 of the offset of the GOT entry of the symbol's TLS descriptor, to be used for general-dynamic TLS accesses.

'hw0_last_tls_gd'

This modifier yields the same value as 'hw0_tls_gd', but it also checks that the value does not overflow.

'hw1_last_tls_gd'

This modifier is used to load bits 16-31 of the offset of the GOT entry of the symbol's TLS descriptor, to be used for general-dynamic TLS accesses. It also checks that the value does not overflow.

'hw0_tls_ie'

This modifier is used to load bits 0-15 of the offset of the GOT entry containing the offset of the symbol's address from the TCB, to be used for initial-exec TLS accesses.

'hw0_last_tls_ie'

This modifier yields the same value as 'hw0_tls_ie', but it also

checks that the value does not overflow.

'hw1_last_tls_ie'

This modifier is used to load bits 16-31 of the offset of the GOT entry containing the offset of the symbol's address from the TCB, to be used for initial-exec TLS accesses. It also checks that the value does not overflow.

'hw0_tls_le'

This modifier is used to load bits 0-15 of the offset of the symbol's address from the TCB, to be used for local-exec TLS accesses.

'hw0_last_tls_le'

This modifier yields the same value as 'hw0_tls_le', but it also checks that the value does not overflow.

'hw1_last_tls_le'

This modifier is used to load bits 16-31 of the offset of the symbol's address from the TCB, to be used for local-exec TLS accesses. It also checks that the value does not overflow.

'tls_gd_call'

This modifier is used to tag an instruction as the "call" part of a calling sequence for a TLS GD reference of its operand.

'tls_gd_add'

This modifier is used to tag an instruction as the "add" part of a calling sequence for a TLS GD reference of its operand.

'tls_ie_load'

This modifier is used to tag an instruction as the "load" part of a calling sequence for a TLS IE reference of its operand.



File: as.info, Node: TILE-Gx Directives, Prev: TILE-Gx Syntax, Up: TILE-Gx-Dependent

9.46.3 TILE-Gx Directives

'**.align** EXPRESSION [, EXPRESSION]'

This is the generic **.ALIGN** directive. The first argument is the requested alignment in bytes.

'**.allow_suspicious_bundles**'

Turns on error checking for combinations of instructions in a bundle that probably indicate a programming error. This is on by default.

'**.no_allow_suspicious_bundles**'

Turns off error checking for combinations of instructions in a

bundle that probably indicate a programming error.

'`.require_canonical_reg_names`'

Require that canonical register names be used, and emit a warning if the numeric names are used. This is on by default.

'`.no_require_canonical_reg_names`'

Permit the use of numeric names for registers that have canonical names.



File: as.info, Node: TILEPro-Dependent, Next: V850-Dependent, Prev: TILE-Gx-Dependent, Up: Machine Dependencies

9.47 TILEPro Dependent Features

=====

* Menu:

* TILEPro Options:: TILEPro Options
* TILEPro Syntax:: TILEPro Syntax
* TILEPro Directives:: TILEPro Directives



File: as.info, Node: TILEPro Options, Next: TILEPro Syntax, Up: TILEPro-Dependent

9.47.1 Options

'as' has no machine-dependent command-line options for TILEPro.



File: as.info, Node: TILEPro Syntax, Next: TILEPro Directives, Prev: TILEPro Options, Up: TILEPro-Dependent

9.47.2 Syntax

Block comments are delimited by `/*` and `*/`. End of line comments may be introduced by `#`.

Instructions consist of a leading opcode or macro name followed by whitespace and an optional comma-separated list of operands:

```
OPCODE [OPERAND, ...]
```

Instructions must be separated by a newline or semicolon.

There are two ways to write code: either write naked instructions, which the assembler is free to combine into VLIW bundles, or specify the VLIW bundles explicitly.

Bundles are specified using curly braces:

```
{ ADD r3,r4,r5 ; ADD r7,r8,r9 ; LW r10,r11 }
```

A bundle can span multiple lines. If you want to put multiple instructions on a line, whether in a bundle or not, you need to separate them with semicolons as in this example.

A bundle may contain one or more instructions, up to the limit specified by the ISA (currently three). If fewer instructions are specified than the hardware supports in a bundle, the assembler inserts 'fnop' instructions automatically.

The assembler will prefer to preserve the ordering of instructions within the bundle, putting the first instruction in a lower-numbered pipeline than the next one, etc. This fact, combined with the optional use of explicit 'fnop' or 'nop' instructions, allows precise control over which pipeline executes each instruction.

If the instructions cannot be bundled in the listed order, the assembler will automatically try to find a valid pipeline assignment. If there is no way to bundle the instructions together, the assembler reports an error.

The assembler does not yet auto-bundle (automatically combine multiple instructions into one bundle), but it reserves the right to do so in the future. If you want to force an instruction to run by itself, put it in a bundle explicitly with curly braces and use 'nop' instructions (not 'fnop') to fill the remaining pipeline slots in that bundle.

* Menu:

* TILEPro Opcodes:: Opcode Naming Conventions.
 * TILEPro Registers:: Register Naming.
 * TILEPro Modifiers:: Symbolic Operand Modifiers.



File: as.info, Node: TILEPro Opcodes, Next: TILEPro Registers, Up: TILEPro Syntax

9.47.2.1 Opcode Names

.....

For a complete list of opcodes and descriptions of their semantics, see 'TILE Processor User Architecture Manual', available upon request at www.tilera.com.



File: as.info, Node: TILEPro Registers, Next: TILEPro Modifiers, Prev: TILEPro Opcodes, Up: TILEPro Syntax

9.47.2.2 Register Names

.....

General-purpose registers are represented by predefined symbols of the form 'rN', where N represents a number between '0' and '63'. However, the following registers have canonical names that must be used instead:

'r54'
 sp

'r55'
 lr

'r56'
 sn

```
'r57'
    idn0

'r58'
    idn1

'r59'
    udn0

'r60'
    udn1

'r61'
    udn2

'r62'
    udn3

'r63'
    zero
```

The assembler will emit a warning if a numeric name is used instead of the canonical name. The `'.no_require_canonical_reg_names'` assembler pseudo-op turns off this warning. `'.require_canonical_reg_names'` turns it back on.



File: as.info, Node: TILEPro Modifiers, Prev: TILEPro Registers, Up: TILEPro Syntax

9.47.2.3 Symbolic Operand Modifiers

.....

The assembler supports several modifiers when using symbol addresses in TILEPro instruction operands. The general syntax is the following:

```
modifier(symbol)
```

The following modifiers are supported:

'lo16'

This modifier is used to load the low 16 bits of the symbol's address, sign-extended to a 32-bit value (sign-extension allows it to be range-checked against signed 16 bit immediate operands without complaint).

'hi16'

This modifier is used to load the high 16 bits of the symbol's address, also sign-extended to a 32-bit value.

'ha16'

'ha16(N)' is identical to 'hi16(N)', except if 'lo16(N)' is negative it adds one to the 'hi16(N)' value. This way 'lo16' and 'ha16' can be added to create any 32-bit value using 'auli'. For example, here is how you move an arbitrary 32-bit address into r3:

```
moveli r3, lo16(sym)
aui r3, r3, ha16(sym)
```

'got'

This modifier is used to load the offset of the GOT entry corresponding to the symbol.

'got_lo16'

This modifier is used to load the sign-extended low 16 bits of the offset of the GOT entry corresponding to the symbol.

'got_hi16'

This modifier is used to load the sign-extended high 16 bits of the offset of the GOT entry corresponding to the symbol.

'got_ha16'

This modifier is like 'got_hi16', but it adds one if 'got_lo16' of the input value is negative.

'plt'

This modifier is used for function symbols. It causes a `_procedure linkage table_`, an array of code stubs, to be created at the time the shared object is created or linked against, together with a global offset table entry. The value is a pc-relative offset to the corresponding stub code in the procedure linkage table. This arrangement causes the run-time symbol resolver to be called to look up and set the value of the symbol the first time the function is called (at latest; depending environment variables). It is only safe to leave the symbol unresolved this way if all references are function calls.

'tls_gd'

This modifier is used to load the offset of the GOT entry of the symbol's TLS descriptor, to be used for general-dynamic TLS accesses.

'tls_gd_lo16'

This modifier is used to load the sign-extended low 16 bits of the offset of the GOT entry of the symbol's TLS descriptor, to be used for general dynamic TLS accesses.

'tls_gd_hi16'

This modifier is used to load the sign-extended high 16 bits of the offset of the GOT entry of the symbol's TLS descriptor, to be used for general dynamic TLS accesses.

'tls_gd_ha16'

This modifier is like 'tls_gd_hi16', but it adds one to the value if 'tls_gd_lo16' of the input value is negative.

'tls_ie'

This modifier is used to load the offset of the GOT entry containing the offset of the symbol's address from the TCB, to be used for initial-exec TLS accesses.

'tls_ie_lo16'

This modifier is used to load the low 16 bits of the offset of the GOT entry containing the offset of the symbol's address from the TCB, to be used for initial-exec TLS accesses.

'tls_ie_hi16'

This modifier is used to load the high 16 bits of the offset of the GOT entry containing the offset of the symbol's address from the TCB, to be used for initial-exec TLS accesses.

'tls_ie_ha16'

This modifier is like 'tls_ie_hi16', but it adds one to the value if 'tls_ie_lo16' of the input value is negative.

'tls_le'

This modifier is used to load the offset of the symbol's address from the TCB, to be used for local-exec TLS accesses.

'tls_le_lo16'

This modifier is used to load the low 16 bits of the offset of the symbol's address from the TCB, to be used for local-exec TLS accesses.

'tls_le_hi16'

This modifier is used to load the high 16 bits of the offset of the symbol's address from the TCB, to be used for local-exec TLS accesses.

'tls_le_ha16'

This modifier is like 'tls_le_hi16', but it adds one to the value if 'tls_le_lo16' of the input value is negative.

'tls_gd_call'

This modifier is used to tag an instruction as the "call" part of a calling sequence for a TLS GD reference of its operand.

'tls_gd_add'

This modifier is used to tag an instruction as the "add" part of a calling sequence for a TLS GD reference of its operand.

'tls_ie_load'

This modifier is used to tag an instruction as the "load" part of a

calling sequence for a TLS IE reference of its operand.



File: as.info, Node: TILEPro Directives, Prev: TILEPro Syntax, Up: TILEPro-Dependent

9.47.3 TILEPro Directives

'`.align EXPRESSION [, EXPRESSION]`'

This is the generic `.ALIGN` directive. The first argument is the requested alignment in bytes.

'`.allow_suspicious_bundles`'

Turns on error checking for combinations of instructions in a bundle that probably indicate a programming error. This is on by default.

'`.no_allow_suspicious_bundles`'

Turns off error checking for combinations of instructions in a bundle that probably indicate a programming error.

'`.require_canonical_reg_names`'

Require that canonical register names be used, and emit a warning if the numeric names are used. This is on by default.

'`.no_require_canonical_reg_names`'

Permit the use of numeric names for registers that have canonical names.



File: as.info, Node: V850-Dependent, Next: Vax-Dependent, Prev: TILEPro-Dependent, Up: Machine Dependencies

9.48 v850 Dependent Features

=====

* Menu:

* V850 Options::	Options
* V850 Syntax::	Syntax
* V850 Floating Point::	Floating Point
* V850 Directives::	V850 Machine Directives
* V850 Opcodes::	Opcodes



File: as.info, Node: V850 Options, Next: V850 Syntax, Up: V850-Dependent

9.48.1 Options

'as' supports the following additional command-line options for the V850 processor family:

'`-wsigned_overflow`'

Causes warnings to be produced when signed immediate values overflow the space available for them within their opcodes. By default this option is disabled as it is possible to receive spurious warnings due to using exact bit patterns as immediate

constants.

'-wunsigned_overflow'

Causes warnings to be produced when unsigned immediate values overflow the space available for them within their opcodes. By default this option is disabled as it is possible to receive spurious warnings due to using exact bit patterns as immediate constants.

'-mv850'

Specifies that the assembled code should be marked as being targeted at the V850 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mv850e'

Specifies that the assembled code should be marked as being targeted at the V850E processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mv850e1'

Specifies that the assembled code should be marked as being targeted at the V850E1 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mv850any'

Specifies that the assembled code should be marked as being targeted at the V850 processor but support instructions that are specific to the extended variants of the process. This allows the production of binaries that contain target specific code, but which are also intended to be used in a generic fashion. For example libgcc.a contains generic routines used by the code produced by GCC for all versions of the v850 architecture, together with support routines only used by the V850E architecture.

'-mv850e2'

Specifies that the assembled code should be marked as being targeted at the V850E2 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mv850e2v3'

Specifies that the assembled code should be marked as being targeted at the V850E2V3 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mv850e2v4'

This is an alias for '-mv850e3v5'.

'-mv850e3v5'

Specifies that the assembled code should be marked as being targeted at the V850E3V5 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

'-mrelax'

Enables relaxation. This allows the `.longcall` and `.longjump` pseudo ops to be used in the assembler source code. These ops label sections of code which are either a long function call or a long branch. The assembler will then flag these sections of code and the linker will attempt to relax them.

'-mgcc-abi'

Marks the generated object file as supporting the old GCC ABI.

'-mrh850-abi'

Marks the generated object file as supporting the RH850 ABI. This is the default.

'-m8byte-align'

Marks the generated object file as supporting a maximum 64-bits of alignment for variables defined in the source code.

'-m4byte-align'

Marks the generated object file as supporting a maximum 32-bits of alignment for variables defined in the source code. This is the default.

'-msoft-float'

Marks the generated object file as not using any floating point instructions - and hence can be linked with other V850 binaries that do or do not use floating point. This is the default for binaries for architectures earlier than the 'e2v3'.

'-mhard-float'

Marks the generated object file as one that uses floating point instructions - and hence can only be linked with other V850 binaries that use the same kind of floating point instructions, or with binaries that do not use floating point at all. This is the default for binaries the 'e2v3' and later architectures.



File: as.info, Node: V850 Syntax, Next: V850 Floating Point, Prev: V850 Options, Up: V850-Dependent

9.48.2 Syntax

* Menu:

* V850-Chars:: Special Characters

* V850-Regs:: Register Names



File: as.info, Node: V850-Chars, Next: V850-Regs, Up: V850 Syntax

9.48.2.1 Special Characters

.....

'#' is the line comment character. If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

Two dashes ('--') can also be used to start a line comment.

The ';' character can be used to separate statements on the same line.



File: as.info, Node: V850-Regs, Prev: V850-Chars, Up: V850 Syntax

9.48.2.2 Register Names

.....

'as' supports the following names for registers:

'general register 0'
r0, zero
'general register 1'
r1
'general register 2'
r2, hp
'general register 3'
r3, sp
'general register 4'
r4, gp
'general register 5'
r5, tp
'general register 6'
r6
'general register 7'
r7
'general register 8'
r8
'general register 9'
r9
'general register 10'
r10
'general register 11'
r11
'general register 12'
r12
'general register 13'
r13
'general register 14'
r14
'general register 15'
r15
'general register 16'
r16
'general register 17'
r17
'general register 18'
r18
'general register 19'
r19
'general register 20'
r20
'general register 21'
r21
'general register 22'
r22
'general register 23'
r23

```
'general register 24'  
  r24  
'general register 25'  
  r25  
'general register 26'  
  r26  
'general register 27'  
  r27  
'general register 28'  
  r28  
'general register 29'  
  r29  
'general register 30'  
  r30, ep  
'general register 31'  
  r31, lp  
'system register 0'  
  eipc  
'system register 1'  
  eipsw  
'system register 2'  
  fepc  
'system register 3'  
  fepsw  
'system register 4'  
  ecr  
'system register 5'  
  psw  
'system register 16'  
  ctpc  
'system register 17'  
  ctpsw  
'system register 18'  
  dbpc  
'system register 19'  
  dbpsw  
'system register 20'  
  ctbp
```



File: as.info, Node: V850 Floating Point, Next: V850 Directives, Prev: V850 Syntax, Up: V850-Dependent

9.48.3 Floating Point

The V850 family uses IEEE floating-point numbers.



File: as.info, Node: V850 Directives, Next: V850 Opcodes, Prev: V850 Floating Point, Up: V850-Dependent

9.48.4 V850 Machine Directives

```
'.offset <EXPRESSION>'
```

Moves the offset into the current section to the specified amount.

```
'.section "name", <type>'
```

This is an extension to the standard `.section` directive. It sets the current section to be `<type>` and creates an alias for this section called "name".

`'.v850'`

Specifies that the assembled code should be marked as being targeted at the V850 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e'`

Specifies that the assembled code should be marked as being targeted at the V850E processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e1'`

Specifies that the assembled code should be marked as being targeted at the V850E1 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e2'`

Specifies that the assembled code should be marked as being targeted at the V850E2 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e2v3'`

Specifies that the assembled code should be marked as being targeted at the V850E2V3 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e2v4'`

Specifies that the assembled code should be marked as being targeted at the V850E3V5 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.

`'.v850e3v5'`

Specifies that the assembled code should be marked as being targeted at the V850E3V5 processor. This allows the linker to detect attempts to link such code with code assembled for other processors.



File: as.info, Node: V850 Opcodes, Prev: V850 Directives, Up: V850-Dependent

9.48.5 Opcodes

'as' implements all the standard V850 opcodes.

'as' also implements the following pseudo ops:

`'hi0()'`

Computes the higher 16 bits of the given expression and stores it into the immediate operand field of the given instruction. For

example:

```
'mulhi hi0(here - there), r5, r6'
```

computes the difference between the address of labels 'here' and 'there', takes the upper 16 bits of this difference, shifts it down 16 bits and then multiplies it by the lower 16 bits in register 5, putting the result into register 6.

'lo()'

Computes the lower 16 bits of the given expression and stores it into the immediate operand field of the given instruction. For example:

```
'addi lo(here - there), r5, r6'
```

computes the difference between the address of labels 'here' and 'there', takes the lower 16 bits of this difference and adds it to register 5, putting the result into register 6.

'hi()'

Computes the higher 16 bits of the given expression and then adds the value of the most significant bit of the lower 16 bits of the expression and stores the result into the immediate operand field of the given instruction. For example the following code can be used to compute the address of the label 'here' and store it into register 6:

```
'movhi hi(here), r0, r6' 'movea lo(here), r6, r6'
```

The reason for this special behaviour is that `movea` performs a sign extension on its immediate operand. So for example if the address of 'here' was `0xFFFFFFFF` then without the special behaviour of the `hi()` pseudo-op the `movhi` instruction would put `0xFFFF0000` into `r6`, then the `movea` instruction would take its immediate operand, `0xFFFF`, sign extend it to 32 bits, `0xFFFFFFFF`, and then add it into `r6` giving `0xFFFEFFFF` which is wrong (the fifth nibble is E). With the `hi()` pseudo op adding in the top bit of the `lo()` pseudo op, the `movhi` instruction actually stores 0 into `r6` (`0xFFFF + 1 = 0x0000`), so that the `movea` instruction stores `0xFFFFFFFF` into `r6` - the right value.

'hilo()'

Computes the 32 bit value of the given expression and stores it into the immediate operand field of the given instruction (which must be a `mov` instruction). For example:

```
'mov hilo(here), r6'
```

computes the absolute address of label 'here' and puts the result into register 6.

'sdaoff()'

Computes the offset of the named variable from the start of the Small Data Area (whose address is held in register 4, the GP register) and stores the result as a 16 bit signed value in the immediate operand field of the given instruction. For example:

```
'ld.w sdaoff(_a_variable)[gp],r6'
```

loads the contents of the location pointed to by the label '_a_variable' into register 6, provided that the label is located somewhere within +/- 32K of the address held in the GP register. [Note the linker assumes that the GP register contains a fixed address set to the address of the label called '__gp'. This can either be set up automatically by the linker, or specifically set by using the '--defsym __gp=<value>' command line option].

'tdaoff()'

Computes the offset of the named variable from the start of the Tiny Data Area (whose address is held in register 30, the EP register) and stores the result as a 4, 5, 7 or 8 bit unsigned value in the immediate operand field of the given instruction. For example:

```
'sld.w tdaoff(_a_variable)[ep],r6'
```

loads the contents of the location pointed to by the label '_a_variable' into register 6, provided that the label is located somewhere within +256 bytes of the address held in the EP register. [Note the linker assumes that the EP register contains a fixed address set to the address of the label called '__ep'. This can either be set up automatically by the linker, or specifically set by using the '--defsym __ep=<value>' command line option].

'zdaoff()'

Computes the offset of the named variable from address 0 and stores the result as a 16 bit signed value in the immediate operand field of the given instruction. For example:

```
'movea zdaoff(_a_variable),zero,r6'
```

puts the address of the label '_a_variable' into register 6, assuming that the label is somewhere within the first 32K of memory. (Strictly speaking it is also possible to access the last 32K of memory as well, as the offsets are signed).

'ctoff()'

Computes the offset of the named variable from the start of the Call Table Area (whose address is held in system register 20, the CTBP register) and stores the result as a 6 or 16 bit unsigned value in the immediate field of the given instruction or piece of data. For example:

```
'callt ctoff(table_func1)'
```

will put the call to the function whose address is held in the call table at the location labeled 'table_func1'.

'longcall name'

Indicates that the following sequence of instructions is a long call to function 'name'. The linker will attempt to shorten this call sequence if 'name' is within a 22bit offset of the call. Only valid if the '-mrelax' command line switch has been enabled.

'longjump name'

Indicates that the following sequence of instructions is a long jump to label 'name'. The linker will attempt to shorten this code

sequence if 'name' is within a 22bit offset of the jump. Only valid if the '-mrelax' command line switch has been enabled.

For information on the V850 instruction set, see 'V850 Family 32-/16-Bit single-Chip Microcontroller Architecture Manual' from NEC. Ltd.

US

File: as.info, Node: Vax-Dependent, Next: Visium-Dependent, Prev: V850-Dependent, Up: Machine Dependencies

9.49 VAX Dependent Features

=====

* Menu:

* VAX-Opts::	VAX Command-Line Options
* VAX-float::	VAX Floating Point
* VAX-directives::	Vax Machine Directives
* VAX-opcodes::	VAX Opcodes
* VAX-branch::	VAX Branch Improvement
* VAX-operands::	VAX Operands
* VAX-no::	Not Supported on VAX
* VAX-Syntax::	VAX Syntax

US

File: as.info, Node: VAX-Opts, Next: VAX-float, Up: Vax-Dependent

9.49.1 VAX Command-Line Options

The Vax version of 'as' accepts any of the following options, gives a warning message that the option was ignored and proceeds. These options are for compatibility with scripts designed for other people's assemblers.

'-D (Debug)'

'-S (Symbol Table)'

'-T (Token Trace)'

These are obsolete options used to debug old assemblers.

'-d (Displacement size for JUMPs)'

This option expects a number following the '-d'. Like options that expect filenames, the number may immediately follow the '-d' (old standard) or constitute the whole of the command line argument that follows '-d' (GNU standard).

'-V (Virtualize Interpass Temporary File)'

Some other assemblers use a temporary file. This option commanded them to keep the information in active memory rather than in a disk file. 'as' always does this, so this option is redundant.

'-J (JUMPify Longer Branches)'

Many 32-bit computers permit a variety of branch instructions to do the same job. Some of these instructions are short (and fast) but have a limited range; others are long (and slow) but can branch anywhere in virtual memory. Often there are 3 flavors of branch: short, medium and long. Some other assemblers would emit short and medium branches, unless told by this option to emit short and long

branches.

'-t (Temporary File Directory)'

Some other assemblers may use a temporary file, and this option takes a filename being the directory to site the temporary file. Since 'as' does not use a temporary disk file, this option makes no difference. '-t' needs exactly one filename.

The Vax version of the assembler accepts additional options when compiled for VMS:

'-h N'

External symbol or section (used for global variables) names are not case sensitive on VAX/VMS and always mapped to upper case. This is contrary to the C language definition which explicitly distinguishes upper and lower case. To implement a standard conforming C compiler, names must be changed (mapped) to preserve the case information. The default mapping is to convert all lower case characters to uppercase and adding an underscore followed by a 6 digit hex value, representing a 24 digit binary value. The one digits in the binary value represent which characters are uppercase in the original symbol name.

The '-h N' option determines how we map names. This takes several values. No '-h' switch at all allows case hacking as described above. A value of zero ('-h0') implies names should be upper case, and inhibits the case hack. A value of 2 ('-h2') implies names should be all lower case, with no case hack. A value of 3 ('-h3') implies that case should be preserved. The value 1 is unused. The '-H' option directs 'as' to display every mapped symbol during assembly.

Symbols whose names include a dollar sign '\$' are exceptions to the general name mapping. These symbols are normally only used to reference VMS library names. Such symbols are always mapped to upper case.

'-+'

The '-+' option causes 'as' to truncate any symbol name larger than 31 characters. The '-+' option also prevents some code following the '_main' symbol normally added to make the object file compatible with Vax-11 "C".

'-1'

This option is ignored for backward compatibility with 'as' version 1.x.

'-H'

The '-H' option causes 'as' to print every symbol which was changed by case mapping.



File: as.info, Node: VAX-float, Next: VAX-directives, Prev: VAX-Opts, Up: Vax-Dependent

9.49.2 VAX Floating Point

Conversion of flonums to floating point is correct, and compatible with

previous assemblers. Rounding is towards zero if the remainder is exactly half the least significant bit.

'D', 'F', 'G' and 'H' floating point formats are understood.

Immediate floating literals (*_e.g._* 'S`\$6.9') are rendered correctly. Again, rounding is towards zero in the boundary case.

The '.float' directive produces 'f' format numbers. The '.double' directive produces 'd' format numbers.

US

File: as.info, Node: VAX-directives, Next: VAX-opcodes, Prev: VAX-float, Up: Vax-Dependent

9.49.3 Vax Machine Directives

The Vax version of the assembler supports four directives for generating Vax floating point constants. They are described in the table below.

'*.dfloat*'

This expects zero or more flonums, separated by commas, and assembles Vax 'd' format 64-bit floating point constants.

'*.ffloat*'

This expects zero or more flonums, separated by commas, and assembles Vax 'f' format 32-bit floating point constants.

'*.gfloat*'

This expects zero or more flonums, separated by commas, and assembles Vax 'g' format 64-bit floating point constants.

'*.hfloat*'

This expects zero or more flonums, separated by commas, and assembles Vax 'h' format 128-bit floating point constants.

US

File: as.info, Node: VAX-opcodes, Next: VAX-branch, Prev: VAX-directives, Up: Vax-Dependent

9.49.4 VAX Opcodes

All DEC mnemonics are supported. Beware that 'case...' instructions have exactly 3 operands. The dispatch table that follows the 'case...' instruction should be made with '.word' statements. This is compatible with all unix assemblers we know of.

US

File: as.info, Node: VAX-branch, Next: VAX-operands, Prev: VAX-opcodes, Up: Vax-Dependent

9.49.5 VAX Branch Improvement

Certain pseudo opcodes are permitted. They are for branch instructions. They expand to the shortest branch instruction that reaches the target. Generally these mnemonics are made by substituting 'j' for 'b' at the

start of a DEC mnemonic. This feature is included both for compatibility and to help compilers. If you do not need this feature, avoid these opcodes. Here are the mnemonics, and the code they can expand into.

```
'jbsb'
  'Jsb' is already an instruction mnemonic, so we chose 'jbsb'.
  (byte displacement)
    'bsbb ...'
  (word displacement)
    'bsbw ...'
  (long displacement)
    'jsb ...'

'jbr'
'jr'
  Unconditional branch.
  (byte displacement)
    'brb ...'
  (word displacement)
    'brw ...'
  (long displacement)
    'jmp ...'

'jCOND'
  COND may be any one of the conditional branches 'neq', 'nequ',
  'eql', 'eqlu', 'gtr', 'geq', 'lss', 'gtru', 'lequ', 'vc', 'vs',
  'gequ', 'cc', 'lssu', 'cs'. COND may also be one of the bit tests
  'bs', 'bc', 'bss', 'bcs', 'bsc', 'bcc', 'bssi', 'bccci', 'lbs',
  'lbc'. NOTCOND is the opposite condition to COND.
  (byte displacement)
    'bCOND ...'
  (word displacement)
    'bNOTCOND foo ; brw ... ; foo:'
  (long displacement)
    'bNOTCOND foo ; jmp ... ; foo:'

'jacbX'
  X may be one of 'b d f g h l w'.
  (word displacement)
    'OPCODE ...'
  (long displacement)
    OPCODE ..., foo ;
    brb bar ;
    foo: jmp ... ;
    bar:

'jaobYYY'
  YYY may be one of 'lss leq'.

'jsobZZZ'
  ZZZ may be one of 'geq gtr'.
  (byte displacement)
    'OPCODE ...'
  (word displacement)
    OPCODE ..., foo ;
    brb bar ;
    foo: brw DESTINATION ;
    bar:
  (long displacement)
    OPCODE ..., foo ;
    brb bar ;
    foo: jmp DESTINATION ;
    bar:
```

```
'aobleq'
'aoblss'
'sobgeq'
'sobgtr'
  (byte displacement)
    'OPCODE ...'
  (word displacement)
    OPCODE ..., foo ;
    brb bar ;
    foo: brw DESTINATION ;
    bar:
  (long displacement)
    OPCODE ..., foo ;
    brb bar ;
    foo: jmp DESTINATION ;
    bar:
```



File: as.info, Node: VAX-operands, Next: VAX-no, Prev: VAX-branch, Up: Vax-Dependent

9.49.6 VAX Operands

The immediate character is '\$' for Unix compatibility, not '#' as DEC writes it.

The indirect character is '*' for Unix compatibility, not '@' as DEC writes it.

The displacement sizing character is '`' (an accent grave) for Unix compatibility, not '^' as DEC writes it. The letter preceding '`' may have either case. 'G' is not understood, but all other letters ('b i l s w') are understood.

Register names understood are 'r0 r1 r2 ... r15 ap fp sp pc'. Upper and lower case letters are equivalent.

For instance
tstb *w`\$4(r5)

Any expression is permitted in an operand. Operands are comma separated.



File: as.info, Node: VAX-no, Next: VAX-Syntax, Prev: VAX-operands, Up: Vax-Dependent

9.49.7 Not Supported on VAX

Vax bit fields can not be assembled with 'as'. Someone can add the required code if they really need it.



File: as.info, Node: VAX-Syntax, Prev: VAX-no, Up: Vax-Dependent

9.49.8 VAX Syntax

* Menu:

* VAX-Chars:: Special Characters



File: as.info, Node: VAX-Chars, Up: VAX-Syntax

9.49.8.1 Special Characters

.....

The presence of a '#' appearing anywhere on a line indicates the start of a comment that extends to the end of that line.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The ';' character can be used to separate statements on the same line.



File: as.info, Node: Visium-Dependent, Next: XGATE-Dependent, Prev: Vax-Dependent, Up: Machine Dependencies

9.50 Visium Dependent Features

=====

* Menu:

* Visium Options:: Options
 * Visium Syntax:: Syntax
 * Visium Opcodes:: Opcodes



File: as.info, Node: Visium Options, Next: Visium Syntax, Up: Visium-Dependent

9.50.1 Options

The Visium assembler implements one machine-specific option:

'-mtune=ARCH'

This option specifies the target architecture. If an attempt is made to assemble an instruction that will not execute on the target architecture, the assembler will issue an error message.

The following names are recognized: 'mcm24' 'mcm' 'gr5' 'gr6'



File: as.info, Node: Visium Syntax, Next: Visium Opcodes, Prev: Visium Options, Up: Visium-Dependent

9.50.2 Syntax

* Menu:

* Visium Characters:: Special Characters
 * Visium Registers:: Register Names



File: as.info, Node: Visium Characters, Next: Visium Registers, Up: Visium Syntax

9.50.2.1 Special Characters

.....

Line comments are introduced either by the '!' character or by the ';' character appearing anywhere on a line.

A hash character ('#') as the first character on a line also marks the start of a line comment, but in this case it could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

The Visium assembler does not currently support a line separator character.



File: as.info, Node: Visium Registers, Prev: Visium Characters, Up: Visium Syntax

9.50.2.2 Register Names

.....

Registers can be specified either by using their canonical mnemonic names or by using their alias if they have one, for example 'sp'.



File: as.info, Node: Visium Opcodes, Prev: Visium Syntax, Up: Visium-Dependent

9.50.3 Opcodes

All the standard opcodes of the architecture are implemented, along with the following three pseudo-instructions: 'cmp', 'cmpc', 'move'.

In addition, the following two illegal opcodes are implemented and used by the simulation:

```
stop    5-bit immediate, SourceA
trace   5-bit immediate, SourceA
```



File: as.info, Node: XGATE-Dependent, Next: XSTORMY16-Dependent, Prev: Visium-Dependent, Up: Machine Dependencies

9.51 XGATE Dependent Features

=====

* Menu:

```
* XGATE-Opts::      XGATE Options
* XGATE-Syntax::   Syntax
* XGATE-Directives:: Assembler Directives
* XGATE-Float::    Floating Point
* XGATE-opcodes::  Opcodes
```



File: as.info, Node: XGATE-Opts, Next: XGATE-Syntax, Up: XGATE-Dependent

9.51.1 XGATE Options

The Freescale XGATE version of 'as' has a few machine dependent options.

'-mshort'

This option controls the ABI and indicates to use a 16-bit integer ABI. It has no effect on the assembled instructions. This is the default.

'-mlong'

This option controls the ABI and indicates to use a 32-bit integer ABI.

'-mshort-double'

This option controls the ABI and indicates to use a 32-bit float ABI. This is the default.

'-mlong-double'

This option controls the ABI and indicates to use a 64-bit float ABI.

'--print-insn-syntax'

You can use the '--print-insn-syntax' option to obtain the syntax description of the instruction when an error is detected.

'--print-opcodes'

The '--print-opcodes' option prints the list of all the instructions with their syntax. Once the list is printed 'as' exits.



File: as.info, Node: XGATE-Syntax, Next: XGATE-Directives, Prev: XGATE-Opts, Up: XGATE-Dependent

9.51.2 Syntax

In XGATE RISC syntax, the instruction name comes first and it may be followed by up to three operands. Operands are separated by commas (','). 'as' will complain if too many operands are specified for a given instruction. The same will happen if you specified too few operands.

```

nop
ldl #23
CMP R1, R2

```

The presence of a ';' character or a '!' character anywhere on a line indicates the start of a comment that extends to the end of that line.

A '*' or a '#' character at the start of a line also introduces a line comment, but these characters do not work elsewhere on the line. If the first character of the line is a '#' then as well as starting a comment, the line could also be logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing:).

The XGATE assembler does not currently support a line separator character.

The following addressing modes are understood for XGATE:

"Inherent"
' '

"Immediate 3 Bit Wide"
'#NUMBER'

"Immediate 4 Bit Wide"
'#NUMBER'

"Immediate 8 Bit Wide"
'#NUMBER'

"Monadic Addressing"
'REG'

"Dyadic Addressing"
'REG, REG'

"Triadic Addressing"
'REG, REG, REG'

"Relative Addressing 9 Bit Wide"
'*SYMBOL'

"Relative Addressing 10 Bit Wide"
'*SYMBOL'

"Index Register plus Immediate Offset"
'REG, (REG, #NUMBER)'

"Index Register plus Register Offset"
'REG, REG, REG'

"Index Register plus Register Offset with Post-increment"
'REG, REG, REG+'

"Index Register plus Register Offset with Pre-decrement"
'REG, REG, -REG'

The register can be either 'R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6' or 'R7'.

Convenience macro opcodes to deal with 16-bit values have been added.

"Immediate 16 Bit Wide"
'#NUMBER', or '*SYMBOL'

For example:

```
ldw R1, #1024
ldw R3, timer
ldw R1, (R1, #0)
COM R1
stw R2, (R1, #0)
```



File: as.info, Node: XGATE-Directives, Next: XGATE-Float, Prev: XGATE-Syntax, Up: XGATE-Dependent

9.51.3 Assembler Directives

The XGATE version of 'as' have the following specific assembler directives:



File: as.info, Node: XGATE-Float, Next: XGATE-opcodes, Prev: XGATE-Directives, Up: XGATE-Dependent

9.51.4 Floating Point

Packed decimal (P) format floating literals are not supported(yet).

The floating point formats generated by directives are these.

'float'

'Single' precision floating point constants.

'double'

'Double' precision floating point constants.

'extend'

'ldouble'

'Extended' precision ('long double') floating point constants.



File: as.info, Node: XGATE-opcodes, Prev: XGATE-Float, Up: XGATE-Dependent

9.51.5 Opcodes



File: as.info, Node: XSTORMY16-Dependent, Next: Xtensa-Dependent, Prev: XGATE-Dependent, Up: Machine Dependencies

9.52 XStormy16 Dependent Features

=====

* Menu:

* XStormy16 Syntax::	Syntax
* XStormy16 Directives::	Machine Directives
* XStormy16 Opcodes::	Pseudo-Opcodes



File: as.info, Node: XStormy16 Syntax, Next: XStormy16 Directives, Up: XSTORMY16-Dependent

9.52.1 Syntax

* Menu:

* XStormy16-Chars:: Special Characters



File: as.info, Node: XStormy16-Chars, Up: XStormy16 Syntax

9.52.1.1 Special Characters

.....

'#' is the line comment character. If a '#' appears as the first character of a line, the whole line is treated as a comment, but in this case the line can also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

A semicolon(';') can be used to start a comment that extends from wherever the character appears on the line up to the end of the line.

The '|' character can be used to separate statements on the same line.



File: as.info, Node: XStormy16 Directives, Next: XStormy16 Opcodes, Prev: XStormy16 Syntax, Up: XSTORMY16-Dependent

9.52.2 XStormy16 Machine Directives

'`.16bit_pointers`'

Like the '`--16bit-pointers`' command line option this directive indicates that the assembly code makes use of 16-bit pointers.

'`.32bit_pointers`'

Like the '`--32bit-pointers`' command line option this directive indicates that the assembly code makes use of 32-bit pointers.

'`.no_pointers`'

Like the '`--no-pointers`' command line option this directive indicates that the assembly code does not makes use pointers.



File: as.info, Node: XStormy16 Opcodes, Prev: XStormy16 Directives, Up: XSTORMY16-Dependent

9.52.3 XStormy16 Pseudo-Opcodes

'`as`' implements all the standard XStormy16 opcodes.

'`as`' also implements the following pseudo ops:

'`@lo()`'

Computes the lower 16 bits of the given expression and stores it into the immediate operand field of the given instruction. For example:

```
'add r6, @lo( here - there )'
```

computes the difference between the address of labels '`here`' and '`there`', takes the lower 16 bits of this difference and adds it to

register 6.

'@hi()'

Computes the higher 16 bits of the given expression and stores it into the immediate operand field of the given instruction. For example:

```
'addc r7, @hi(here - there)'
```

computes the difference between the address of labels 'here' and 'there', takes the upper 16 bits of this difference, shifts it down 16 bits and then adds it, along with the carry bit, to the value in register 7.

US

File: as.info, Node: Xtensa-Dependent, Next: Z80-Dependent, Prev: XSTORMY16-Dependent, Up: Machine Dependencies

9.53 Xtensa Dependent Features

=====

This chapter covers features of the GNU assembler that are specific to the Xtensa architecture. For details about the Xtensa instruction set, please consult the 'Xtensa Instruction Set Architecture (ISA) Reference Manual'.

* Menu:

* Xtensa Options::	Command-line Options.
* Xtensa Syntax::	Assembler Syntax for Xtensa Processors.
* Xtensa Optimizations::	Assembler Optimizations.
* Xtensa Relaxation::	Other Automatic Transformations.
* Xtensa Directives::	Directives for Xtensa Processors.

US

File: as.info, Node: Xtensa Options, Next: Xtensa Syntax, Up: Xtensa-Dependent

9.53.1 Command Line Options

'--text-section-literals | --no-text-section-literals'

Control the treatment of literal pools. The default is '--no-text-section-literals', which places literals in separate sections in the output file. This allows the literal pool to be placed in a data RAM/ROM. With '--text-section-literals', the literals are interspersed in the text section in order to keep them as close as possible to their references. This may be necessary for large assembly files, where the literals would otherwise be out of range of the 'L32R' instructions in the text section. Literals are grouped into pools following '.literal_position' directives or preceding 'ENTRY' instructions. These options only affect literals referenced via PC-relative 'L32R' instructions; literals for absolute mode 'L32R' instructions are handled separately. *Note literal: Literal Directive.

'--auto-litpools | --no-auto-litpools'

Control the treatment of literal pools. The default is '--no-auto-litpools', which in the absence of '--text-section-literals' places literals in separate sections in

the output file. This allows the literal pool to be placed in a data RAM/ROM. With '--auto-litpools', the literals are interspersed in the text section in order to keep them as close as possible to their references, explicit '.literal_position' directives are not required. This may be necessary for very large functions, where single literal pool at the beginning of the function may not be reachable by 'L32R' instructions at the end. These options only affect literals referenced via PC-relative 'L32R' instructions; literals for absolute mode 'L32R' instructions are handled separately. When used together with '--text-section-literals', '--auto-litpools' takes precedence. *Note literal: Literal Directive.

'--absolute-literals | --no-absolute-literals'

Indicate to the assembler whether 'L32R' instructions use absolute or PC-relative addressing. If the processor includes the absolute addressing option, the default is to use absolute 'L32R' relocations. Otherwise, only the PC-relative 'L32R' relocations can be used.

'--target-align | --no-target-align'

Enable or disable automatic alignment to reduce branch penalties at some expense in code size. *Note Automatic Instruction Alignment: Xtensa Automatic Alignment. This optimization is enabled by default. Note that the assembler will always align instructions like 'LOOP' that have fixed alignment requirements.

'--longcalls | --no-longcalls'

Enable or disable transformation of call instructions to allow calls across a greater range of addresses. *Note Function Call Relaxation: Xtensa Call Relaxation. This option should be used when call targets can potentially be out of range. It may degrade both code size and performance, but the linker can generally optimize away the unnecessary overhead when a call ends up within range. The default is '--no-longcalls'.

'--transform | --no-transform'

Enable or disable all assembler transformations of Xtensa instructions, including both relaxation and optimization. The default is '--transform'; '--no-transform' should only be used in the rare cases when the instructions must be exactly as specified in the assembly source. Using '--no-transform' causes out of range instruction operands to be errors.

'--rename-section OLDNAME=NEWNAME'

Rename the OLDNAME section to NEWNAME. This option can be used multiple times to rename multiple sections.

'--trampolines | --no-trampolines'

Enable or disable transformation of jump instructions to allow jumps across a greater range of addresses. *Note Jump Trampolines: Xtensa Jump Relaxation. This option should be used when jump targets can potentially be out of range. In the absence of such jumps this option does not affect code size or performance. The default is '--trampolines'.



9.53.2 Assembler Syntax

Block comments are delimited by `'/*'` and `'*/'`. End of line comments may be introduced with either `'#'` or `'//'`.

If a `'#'` appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (`*note Comments::`) or a preprocessor control command (`*note Preprocessing::`).

Instructions consist of a leading opcode or macro name followed by whitespace and an optional comma-separated list of operands:

```
OPCODE [OPERAND, ...]
```

Instructions must be separated by a newline or semicolon (`';'`).

FLIX instructions, which bundle multiple opcodes together in a single instruction, are specified by enclosing the bundled opcodes inside braces:

```
{
  [FORMAT]
  OPCODE0 [OPERANDS]
  OPCODE1 [OPERANDS]
  OPCODE2 [OPERANDS]
  ...
}
```

The opcodes in a FLIX instruction are listed in the same order as the corresponding instruction slots in the TIE format declaration. Directives and labels are not allowed inside the braces of a FLIX instruction. A particular TIE format name can optionally be specified immediately after the opening brace, but this is usually unnecessary. The assembler will automatically search for a format that can encode the specified opcodes, so the format name need only be specified in rare cases where there is more than one applicable format and where it matters which of those formats is used. A FLIX instruction can also be specified on a single line by separating the opcodes with semicolons:

```
{ [FORMAT;] OPCODE0 [OPERANDS]; OPCODE1 [OPERANDS]; OPCODE2 [OPERANDS]; ... }
```

If an opcode can only be encoded in a FLIX instruction but is not specified as part of a FLIX bundle, the assembler will choose the smallest format where the opcode can be encoded and will fill unused instruction slots with no-ops.

* Menu:

```
* Xtensa Opcodes::          Opcode Naming Conventions.
* Xtensa Registers::       Register Naming.
```



File: as.info, Node: Xtensa Opcodes, Next: Xtensa Registers, Up: Xtensa Syntax

9.53.2.1 Opcode Names

.....

See the 'Xtensa Instruction Set Architecture (ISA) Reference Manual' for a complete list of opcodes and descriptions of their semantics.

If an opcode name is prefixed with an underscore character ('_'), 'as' will not transform that instruction in any way. The underscore prefix disables both optimization (*note Xtensa Optimizations: Xtensa Optimizations.) and relaxation (*note Xtensa Relaxation: Xtensa Relaxation.) for that particular instruction. Only use the underscore prefix when it is essential to select the exact opcode produced by the assembler. Using this feature unnecessarily makes the code less efficient by disabling assembler optimization and less flexible by disabling relaxation.

Note that this special handling of underscore prefixes only applies to Xtensa opcodes, not to either built-in macros or user-defined macros. When an underscore prefix is used with a macro (e.g., '_MOV'), it refers to a different macro. The assembler generally provides built-in macros both with and without the underscore prefix, where the underscore versions behave as if the underscore carries through to the instructions in the macros. For example, '_MOV' may expand to '_MOV.N'.

The underscore prefix only applies to individual instructions, not to series of instructions. For example, if a series of instructions have underscore prefixes, the assembler will not transform the individual instructions, but it may insert other instructions between them (e.g., to align a 'LOOP' instruction). To prevent the assembler from modifying a series of instructions as a whole, use the 'no-transform' directive.
*Note transform: Transform Directive.



File: as.info, Node: Xtensa Registers, Prev: Xtensa Opcodes, Up: Xtensa Syntax

9.53.2.2 Register Names

.....

The assembly syntax for a register file entry is the "short" name for a TIE register file followed by the index into that register file. For example, the general-purpose 'AR' register file has a short name of 'a', so these registers are named 'a0'...'a15'. As a special feature, 'sp' is also supported as a synonym for 'a1'. Additional registers may be added by processor configuration options and by designer-defined TIE extensions. An initial '\$' character is optional in all register names.



File: as.info, Node: Xtensa Optimizations, Next: Xtensa Relaxation, Prev: Xtensa Syntax, Up: Xtensa-Dependent

9.53.3 Xtensa Optimizations

The optimizations currently supported by 'as' are generation of density instructions where appropriate and automatic branch target alignment.

* Menu:

- * Density Instructions:: Using Density Instructions.
- * Xtensa Automatic Alignment:: Automatic Instruction Alignment.



File: as.info, Node: Density Instructions, Next: Xtensa Automatic Alignment, Up: Xtensa Optimizations

9.53.3.1 Using Density Instructions

.....

The Xtensa instruction set has a code density option that provides 16-bit versions of some of the most commonly used opcodes. Use of these opcodes can significantly reduce code size. When possible, the assembler automatically translates instructions from the core Xtensa instruction set into equivalent instructions from the Xtensa code density option. This translation can be disabled by using underscore prefixes (*note Opcode Names: Xtensa Opcodes.), by using the '--no-transform' command-line option (*note Command Line Options: Xtensa Options.), or by using the 'no-transform' directive (*note transform: Transform Directive.).

It is a good idea not to use the density instructions directly. The assembler will automatically select dense instructions where possible. If you later need to use an Xtensa processor without the code density option, the same assembly code will then work without modification.



File: as.info, Node: Xtensa Automatic Alignment, Prev: Density Instructions, Up: Xtensa Optimizations

9.53.3.2 Automatic Instruction Alignment

.....

The Xtensa assembler will automatically align certain instructions, both to optimize performance and to satisfy architectural requirements.

As an optimization to improve performance, the assembler attempts to align branch targets so they do not cross instruction fetch boundaries. (Xtensa processors can be configured with either 32-bit or 64-bit instruction fetch widths.) An instruction immediately following a call is treated as a branch target in this context, because it will be the target of a return from the call. This alignment has the potential to reduce branch penalties at some expense in code size. This optimization is enabled by default. You can disable it with the '--no-target-align' command-line option (*note Command Line Options: Xtensa Options.).

The target alignment optimization is done without adding instructions that could increase the execution time of the program. If there are density instructions in the code preceding a target, the assembler can change the target alignment by widening some of those instructions to the equivalent 24-bit instructions. Extra bytes of padding can be inserted immediately following unconditional jump and return instructions. This approach is usually successful in aligning many, but not all, branch targets.

The 'LOOP' family of instructions must be aligned such that the first instruction in the loop body does not cross an instruction fetch boundary (e.g., with a 32-bit fetch width, a 'LOOP' instruction must be on either a 1 or 2 mod 4 byte boundary). The assembler knows about this restriction and inserts the minimal number of 2 or 3 byte no-op instructions to satisfy it. When no-op instructions are added, any

label immediately preceding the original loop will be moved in order to refer to the loop instruction, not the newly generated no-op instruction. To preserve binary compatibility across processors with different fetch widths, the assembler conservatively assumes a 32-bit fetch width when aligning 'LOOP' instructions (except if the first instruction in the loop is a 64-bit instruction).

Previous versions of the assembler automatically aligned 'ENTRY' instructions to 4-byte boundaries, but that alignment is now the programmer's responsibility.



File: as.info, Node: Xtensa Relaxation, Next: Xtensa Directives, Prev: Xtensa Optimizations, Up: Xtensa-Dependent

9.53.4 Xtensa Relaxation

When an instruction operand is outside the range allowed for that particular instruction field, 'as' can transform the code to use a functionally-equivalent instruction or sequence of instructions. This process is known as "relaxation". This is typically done for branch instructions because the distance of the branch targets is not known until assembly-time. The Xtensa assembler offers branch relaxation and also extends this concept to function calls, 'MOVI' instructions and other instructions with immediate fields.

* Menu:

- * Xtensa Branch Relaxation:: Relaxation of Branches.
- * Xtensa Call Relaxation:: Relaxation of Function Calls.
- * Xtensa Jump Relaxation:: Relaxation of Jumps.
- * Xtensa Immediate Relaxation:: Relaxation of other Immediate Fields.



File: as.info, Node: Xtensa Branch Relaxation, Next: Xtensa Call Relaxation, Up: Xtensa Relaxation

9.53.4.1 Conditional Branch Relaxation

.....

When the target of a branch is too far away from the branch itself, i.e., when the offset from the branch to the target is too large to fit in the immediate field of the branch instruction, it may be necessary to replace the branch with a branch around a jump. For example,

```
    beqz    a2, L
```

may result in:

```
    bnez.n a2, M
    j L
```

M:

(The 'BNEZ.N' instruction would be used in this example only if the density option is available. Otherwise, 'BNEZ' would be used.)

This relaxation works well because the unconditional jump instruction has a much larger offset range than the various conditional branches.

However, an error will occur if a branch target is beyond the range of a jump instruction. 'as' cannot relax unconditional jumps. Similarly, an error will occur if the original input contains an unconditional jump to a target that is out of range.

Branch relaxation is enabled by default. It can be disabled by using underscore prefixes (*note Opcode Names: Xtensa Opcodes.), the '--no-transform' command-line option (*note Command Line Options: Xtensa Options.), or the 'no-transform' directive (*note transform: Transform Directive.).

US

File: as.info, Node: Xtensa Call Relaxation, Next: Xtensa Jump Relaxation, Prev: Xtensa Branch Relaxation, Up: Xtensa Relaxation

9.53.4.2 Function Call Relaxation

.....

Function calls may require relaxation because the Xtensa immediate call instructions ('CALL0', 'CALL4', 'CALL8' and 'CALL12') provide a PC-relative offset of only 512 Kbytes in either direction. For larger programs, it may be necessary to use indirect calls ('CALLX0', 'CALLX4', 'CALLX8' and 'CALLX12') where the target address is specified in a register. The Xtensa assembler can automatically relax immediate call instructions into indirect call instructions. This relaxation is done by loading the address of the called function into the callee's return address register and then using a 'CALLX' instruction. So, for example:

```
call8 func
```

might be relaxed to:

```
.literal .L1, func
l32r    a8, .L1
callx8  a8
```

Because the addresses of targets of function calls are not generally known until link-time, the assembler must assume the worst and relax all the calls to functions in other source files, not just those that really will be out of range. The linker can recognize calls that were unnecessarily relaxed, and it will remove the overhead introduced by the assembler for those cases where direct calls are sufficient.

Call relaxation is disabled by default because it can have a negative effect on both code size and performance, although the linker can usually eliminate the unnecessary overhead. If a program is too large and some of the calls are out of range, function call relaxation can be enabled using the '--longcalls' command-line option or the 'longcalls' directive (*note longcalls: Longcalls Directive.).

US

File: as.info, Node: Xtensa Jump Relaxation, Next: Xtensa Immediate Relaxation, Prev: Xtensa Call Relaxation, Up: Xtensa Relaxation

9.53.4.3 Jump Relaxation

.....

Jump instruction may require relaxation because the Xtensa jump instruction ('J') provide a PC-relative offset of only 128 Kbytes in

either direction. One option is to use jump long ('J.L') instruction, which depending on jump distance may be assembled as jump ('J') or indirect jump ('JX'). However it needs a free register. When there's no spare register it is possible to plant intermediate jump sites (trampolines) between the jump instruction and its target. These sites may be located in areas unreachable by normal code execution flow, in that case they only contain intermediate jumps, or they may be inserted in the middle of code block, in which case there's an additional jump from the beginning of the trampoline to the instruction past its end. So, for example:

```

    j 1f
    ...
    retw
    ...
    mov a10, a2
    call8 func
    ...
1:
    ...

```

might be relaxed to:

```

    j .LO_TR_1
    ...
    retw
.LO_TR_1:
    j 1f
    ...
    mov a10, a2
    call8 func
    ...
1:
    ...

```

or to:

```

    j .LO_TR_1
    ...
    retw
    ...
    mov a10, a2
    j .LO_TR_0
.LO_TR_1:
    j 1f
.LO_TR_0:
    call8 func
    ...
1:
    ...

```

The Xtensa assembler uses trampolines with jump around only when it cannot find suitable unreachable trampoline. There may be multiple trampolines between the jump instruction and its target.

This relaxation does not apply to jumps to undefined symbols, assuming they will reach their targets once resolved.

Jump relaxation is enabled by default because it does not affect code

size or performance while the code itself is small. This relaxation may be disabled completely with '--no-trampolines' or '--no-transform' command-line options (*note Command Line Options: Xtensa Options.).

US

File: as.info, Node: Xtensa Immediate Relaxation, Prev: Xtensa Jump Relaxation, Up: Xtensa Relaxation

9.53.4.4 Other Immediate Field Relaxation

.....

The assembler normally performs the following other relaxations. They can be disabled by using underscore prefixes (*note Opcode Names: Xtensa Opcodes.), the '--no-transform' command-line option (*note Command Line Options: Xtensa Options.), or the 'no-transform' directive (*note transform: Transform Directive.).

The 'MOVI' machine instruction can only materialize values in the range from -2048 to 2047. Values outside this range are best materialized with 'L32R' instructions. Thus:

```
movi a0, 100000
```

is assembled into the following machine code:

```
.literal .L1, 100000
l32r a0, .L1
```

The 'L8UI' machine instruction can only be used with immediate offsets in the range from 0 to 255. The 'L16SI' and 'L16UI' machine instructions can only be used with offsets from 0 to 510. The 'L32I' machine instruction can only be used with offsets from 0 to 1020. A load offset outside these ranges can be materialized with an 'L32R' instruction if the destination register of the load is different than the source address register. For example:

```
l32i a1, a0, 2040
```

is translated to:

```
.literal .L1, 2040
l32r a1, .L1
add a1, a0, a1
l32i a1, a1, 0
```

If the load destination and source address register are the same, an out-of-range offset causes an error.

The Xtensa 'ADDI' instruction only allows immediate operands in the range from -128 to 127. There are a number of alternate instruction sequences for the 'ADDI' operation. First, if the immediate is 0, the 'ADDI' will be turned into a 'MOV.N' instruction (or the equivalent 'OR' instruction if the code density option is not available). If the 'ADDI' immediate is outside of the range -128 to 127, but inside the range -32896 to 32639, an 'ADDMI' instruction or 'ADDMI'/'ADDI' sequence will be used. Finally, if the immediate is outside of this range and a free register is available, an 'L32R'/'ADD' sequence will be used with a literal allocated from the literal pool.

For example:

```
addi    a5, a6, 0
addi    a5, a6, 512
addi    a5, a6, 513
addi    a5, a6, 50000
```

is assembled into the following:

```
.literal .L1, 50000
mov.n    a5, a6
addmi    a5, a6, 0x200
addmi    a5, a6, 0x200
addi     a5, a5, 1
l32r    a5, .L1
add      a5, a6, a5
```



File: as.info, Node: Xtensa Directives, Prev: Xtensa Relaxation, Up: Xtensa-Dependent

9.53.5 Directives

The Xtensa assembler supports a region-based directive syntax:

```
.begin DIRECTIVE [OPTIONS]
...
.end DIRECTIVE
```

All the Xtensa-specific directives that apply to a region of code use this syntax.

The directive applies to code between the `.begin` and the `.end`. The state of the option after the `.end` reverts to what it was before the `.begin`. A nested `.begin/.end` region can further change the state of the directive without having to be aware of its outer state. For example, consider:

```
.begin no-transform
L:  add a0, a1, a2
    .begin transform
M:  add a0, a1, a2
    .end transform
N:  add a0, a1, a2
    .end no-transform
```

The `'ADD'` opcodes at `'L'` and `'N'` in the outer `'no-transform'` region both result in `'ADD'` machine instructions, but the assembler selects an `'ADD.N'` instruction for the `'ADD'` at `'M'` in the inner `'transform'` region.

The advantage of this style is that it works well inside macros which can preserve the context of their callers.

The following directives are available:

* Menu:

* Schedule Directive:: Enable instruction scheduling.

- * Longcalls Directive:: Use Indirect Calls for Greater Range.
- * Transform Directive:: Disable All Assembler Transformations.
- * Literal Directive:: Intermix Literals with Instructions.
- * Literal Position Directive:: Specify Inline Literal Pool Locations.
- * Literal Prefix Directive:: Specify Literal Section Name Prefix.
- * Absolute Literals Directive:: Control PC-Relative vs. Absolute Literals.



File: as.info, Node: Schedule Directive, Next: Longcalls Directive, Up: Xtensa Directives

9.53.5.1 schedule

.....

The 'schedule' directive is recognized only for compatibility with Tensilica's assembler.

```
.begin [no-]schedule
.end [no-]schedule
```

This directive is ignored and has no effect on 'as'.



File: as.info, Node: Longcalls Directive, Next: Transform Directive, Prev: Schedule Directive, Up: Xtensa Directives

9.53.5.2 longcalls

.....

The 'longcalls' directive enables or disables function call relaxation.
*Note Function Call Relaxation: Xtensa Call Relaxation.

```
.begin [no-]longcalls
.end [no-]longcalls
```

Call relaxation is disabled by default unless the '--longcalls' command-line option is specified. The 'longcalls' directive overrides the default determined by the command-line options.



File: as.info, Node: Transform Directive, Next: Literal Directive, Prev: Longcalls Directive, Up: Xtensa Directives

9.53.5.3 transform

.....

This directive enables or disables all assembler transformation, including relaxation (*note Xtensa Relaxation: Xtensa Relaxation.) and optimization (*note Xtensa Optimizations: Xtensa Optimizations.).

```
.begin [no-]transform
.end [no-]transform
```

Transformations are enabled by default unless the '--no-transform' option is used. The 'transform' directive overrides the default determined by the command-line options. An underscore opcode prefix, disabling transformation of that opcode, always takes precedence over both directives and command-line flags.



File: as.info, Node: Literal Directive, Next: Literal Position Directive, Prev: Transform Directive, Up: Xtensa Directives

9.53.5.4 literal

.....

The `.literal` directive is used to define literal pool data, i.e., read-only 32-bit data accessed via `'L32R'` instructions.

```
.literal LABEL, VALUE[, VALUE...]
```

This directive is similar to the standard `.word` directive, except that the actual location of the literal data is determined by the assembler and linker, not by the position of the `.literal` directive. Using this directive gives the assembler freedom to locate the literal data in the most appropriate place and possibly to combine identical literals. For example, the code:

```
entry sp, 40
.literal .L1, sym
l32r    a4, .L1
```

can be used to load a pointer to the symbol `'sym'` into register `'a4'`. The value of `'sym'` will not be placed between the `'ENTRY'` and `'L32R'` instructions; instead, the assembler puts the data in a literal pool.

Literal pools are placed by default in separate literal sections; however, when using the `'--text-section-literals'` option (*note Command Line Options: Xtensa Options.), the literal pools for PC-relative mode `'L32R'` instructions are placed in the current section.(1) These text section literal pools are created automatically before `'ENTRY'` instructions and manually after `.literal_position` directives (*note `literal_position`: Literal Position Directive.). If there are no preceding `'ENTRY'` instructions, explicit `.literal_position` directives must be used to place the text section literal pools; otherwise, `'as'` will report an error.

When literals are placed in separate sections, the literal section names are derived from the names of the sections where the literals are defined. The base literal section names are `.literal` for PC-relative mode `'L32R'` instructions and `.lit4` for absolute mode `'L32R'` instructions (*note `absolute-literals`: Absolute Literals Directive.). These base names are used for literals defined in the default `.text` section. For literals defined in other sections or within the scope of a `literal_prefix` directive (*note `literal_prefix`: Literal Prefix Directive.), the following rules determine the literal section name:

1. If the current section is a member of a section group, the literal section name includes the group name as a suffix to the base `.literal` or `.lit4` name, with a period to separate the base name and group name. The literal section is also made a member of the group.
2. If the current section name (or `literal_prefix` value) begins with `gnu.linkonce.KIND.`, the literal section name is formed by replacing `gnu.KIND` with the base `.literal` or `.lit4` name. For example, for literals defined in a section named `gnu.linkonce.t.func`, the literal section will be

`'.gnu.linkonce.literal.func'` or `'.gnu.linkonce.lit4.func'`.

3. If the current section name (or `'literal_prefix'` value) ends with `'.text'`, the literal section name is formed by replacing that suffix with the base `'.literal'` or `'.lit4'` name. For example, for literals defined in a section named `'.iram0.text'`, the literal section will be `'.iram0.literal'` or `'.iram0.lit4'`.
4. If none of the preceding conditions apply, the literal section name is formed by adding the base `'.literal'` or `'.lit4'` name as a suffix to the current section name (or `'literal_prefix'` value).

----- Footnotes -----

(1) Literals for the `'.init'` and `'.fini'` sections are always placed in separate sections, even when `'--text-section-literals'` is enabled.



File: as.info, Node: Literal Position Directive, Next: Literal Prefix Directive, Prev: Literal Directive, Up: Xtensa Directives

9.53.5.5 `literal_position`

.....

When using `'--text-section-literals'` to place literals inline in the section being assembled, the `'.literal_position'` directive can be used to mark a potential location for a literal pool.

```
.literal_position
```

The `'.literal_position'` directive is ignored when the `'--text-section-literals'` option is not used or when `'L32R'` instructions use the absolute addressing mode.

The assembler will automatically place text section literal pools before `'ENTRY'` instructions, so the `'.literal_position'` directive is only needed to specify some other location for a literal pool. You may need to add an explicit jump instruction to skip over an inline literal pool.

For example, an interrupt vector does not begin with an `'ENTRY'` instruction so the assembler will be unable to automatically find a good place to put a literal pool. Moreover, the code for the interrupt vector must be at a specific starting address, so the literal pool cannot come before the start of the code. The literal pool for the vector must be explicitly positioned in the middle of the vector (before any uses of the literals, due to the negative offsets used by PC-relative `'L32R'` instructions). The `'.literal_position'` directive can be used to do this. In the following code, the literal for `'M'` will automatically be aligned correctly and is placed after the unconditional jump.

```
.global M
code_start:
    j continue
    .literal_position
    .align 4
continue:
    movi    a4, M
```

US

File: as.info, Node: Literal Prefix Directive, Next: Absolute Literals Directive, Prev: Literal Position Directive, Up: Xtensa Directives

9.53.5.6 literal_prefix

.....

The 'literal_prefix' directive allows you to override the default literal section names, which are derived from the names of the sections where the literals are defined.

```
.begin literal_prefix [NAME]
.end literal_prefix
```

For literals defined within the delimited region, the literal section names are derived from the NAME argument instead of the name of the current section. The rules used to derive the literal section names do not change. *Note literal: Literal Directive. If the NAME argument is omitted, the literal sections revert to the defaults. This directive has no effect when using the '--text-section-literals' option (*note Command Line Options: Xtensa Options.).

US

File: as.info, Node: Absolute Literals Directive, Prev: Literal Prefix Directive, Up: Xtensa Directives

9.53.5.7 absolute-literals

.....

The 'absolute-literals' and 'no-absolute-literals' directives control the absolute vs. PC-relative mode for 'L32R' instructions. These are relevant only for Xtensa configurations that include the absolute addressing option for 'L32R' instructions.

```
.begin [no-]absolute-literals
.end [no-]absolute-literals
```

These directives do not change the 'L32R' mode--they only cause the assembler to emit the appropriate kind of relocation for 'L32R' instructions and to place the literal values in the appropriate section. To change the 'L32R' mode, the program must write the 'LITBASE' special register. It is the programmer's responsibility to keep track of the mode and indicate to the assembler which mode is used in each region of code.

If the Xtensa configuration includes the absolute 'L32R' addressing option, the default is to assume absolute 'L32R' addressing unless the '--no-absolute-literals' command-line option is specified. Otherwise, the default is to assume PC-relative 'L32R' addressing. The 'absolute-literals' directive can then be used to override the default determined by the command-line options.

US

File: as.info, Node: Z80-Dependent, Next: Z8000-Dependent, Prev: Xtensa-Dependent, Up: Machine Dependencies

9.54 Z80 Dependent Features

=====

* Menu:

```

* Z80 Options::          Options
* Z80 Syntax::          Syntax
* Z80 Floating Point::  Floating Point
* Z80 Directives::      Z80 Machine Directives
* Z80 Opcodes::         Opcodes

```



File: as.info, Node: Z80 Options, Next: Z80 Syntax, Up: Z80-Dependent

9.54.1 Options

The Zilog Z80 and Ascii R800 version of 'as' have a few machine dependent options.

```

'-z80'
    Produce code for the Z80 processor.  There are additional options
    to request warnings and error messages for undocumented
    instructions.
'-ignore-undocumented-instructions'
'-Wnud'
    Silently assemble undocumented Z80-instructions that have been
    adopted as documented R800-instructions.
'-ignore-unportable-instructions'
'-Wnup'
    Silently assemble all undocumented Z80-instructions.
'-warn-undocumented-instructions'
'-Wud'
    Issue warnings for undocumented Z80-instructions that work on R800,
    do not assemble other undocumented instructions without warning.
'-warn-unportable-instructions'
'-Wup'
    Issue warnings for other undocumented Z80-instructions, do not
    treat any undocumented instructions as errors.
'-forbid-undocumented-instructions'
'-Fud'
    Treat all undocumented z80-instructions as errors.
'-forbid-unportable-instructions'
'-Fup'
    Treat undocumented z80-instructions that do not work on R800 as
    errors.

'-r800'
    Produce code for the R800 processor.  The assembler does not
    support undocumented instructions for the R800.  In line with
    common practice, 'as' uses Z80 instruction names for the R800
    processor, as far as they exist.

```



File: as.info, Node: Z80 Syntax, Next: Z80 Floating Point, Prev: Z80 Options, Up: Z80-Dependent

9.54.2 Syntax

The assembler syntax closely follows the 'Z80 family CPU User Manual' by Zilog. In expressions a single '=' may be used as "is equal to"

comparison operator.

Suffices can be used to indicate the radix of integer constants; 'H' or 'h' for hexadecimal, 'D' or 'd' for decimal, 'Q', 'O', 'q' or 'o' for octal, and 'B' for binary.

The suffix 'b' denotes a backreference to local label.

* Menu:

```
* Z80-Chars::          Special Characters
* Z80-Regs::          Register Names
* Z80-Case::          Case Sensitivity
```



File: as.info, Node: Z80-Chars, Next: Z80-Regs, Up: Z80 Syntax

9.54.2.1 Special Characters

.....

The semicolon ';' is the line comment character;

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments:.) or a preprocessor control command (*note Preprocessing:.).

The Z80 assembler does not support a line separator character.

The dollar sign '\$' can be used as a prefix for hexadecimal numbers and as a symbol denoting the current location counter.

A backslash '\' is an ordinary character for the Z80 assembler.

The single quote ''' must be followed by a closing quote. If there is one character in between, it is a character constant, otherwise it is a string constant.



File: as.info, Node: Z80-Regs, Next: Z80-Case, Prev: Z80-Chars, Up: Z80 Syntax

9.54.2.2 Register Names

.....

The registers are referred to with the letters assigned to them by Zilog. In addition 'as' recognizes 'ixl' and 'ixh' as the least and most significant octet in 'ix', and similarly 'iyl' and 'iyh' as parts of 'iy'.



File: as.info, Node: Z80-Case, Prev: Z80-Regs, Up: Z80 Syntax

9.54.2.3 Case Sensitivity

.....

Upper and lower case are equivalent in register names, opcodes, condition codes and assembler directives. The case of letters is significant in labels and symbol names. The case is also important to distinguish the suffix 'b' for a backward reference to a local label

from the suffix 'B' for a number in binary notation.



File: as.info, Node: Z80 Floating Point, Next: Z80 Directives, Prev: Z80 Syntax,
Up: Z80-Dependent

9.54.3 Floating Point

Floating-point numbers are not supported.



File: as.info, Node: Z80 Directives, Next: Z80 Opcodes, Prev: Z80 Floating Point,
Up: Z80-Dependent

9.54.4 Z80 Assembler Directives

'as' for the Z80 supports some additional directives for compatibility with other assemblers.

These are the additional directives in 'as' for the Z80:

'db EXPRESSION|STRING[,EXPRESSION|STRING...]'

'defb EXPRESSION|STRING[,EXPRESSION|STRING...]'

For each STRING the characters are copied to the object file, for each other EXPRESSION the value is stored in one byte. A warning is issued in case of an overflow.

'dw EXPRESSION[,EXPRESSION...]'

'defw EXPRESSION[,EXPRESSION...]'

For each EXPRESSION the value is stored in two bytes, ignoring overflow.

'd24 EXPRESSION[,EXPRESSION...]'

'def24 EXPRESSION[,EXPRESSION...]'

For each EXPRESSION the value is stored in three bytes, ignoring overflow.

'd32 EXPRESSION[,EXPRESSION...]'

'def32 EXPRESSION[,EXPRESSION...]'

For each EXPRESSION the value is stored in four bytes, ignoring overflow.

'ds COUNT[, VALUE]'

'defs COUNT[, VALUE]'

Fill COUNT bytes in the object file with VALUE, if VALUE is omitted it defaults to zero.

'SYMBOL equ EXPRESSION'

'SYMBOL defl EXPRESSION'

These directives set the value of SYMBOL to EXPRESSION. If 'equ' is used, it is an error if SYMBOL is already defined. Symbols defined with 'equ' are not protected from redefinition.

'set'

This is a normal instruction on Z80, and not an assembler directive.

'psect NAME'

A synonym for *Note Section::, no second argument should be given.



File: as.info, Node: Z80 Opcodes, Prev: Z80 Directives, Up: Z80-Dependent

9.54.5 Opcodes

In line with common practice, Z80 mnemonics are used for both the Z80 and the R800.

In many instructions it is possible to use one of the half index registers ('ixl','ixh','iyl','iyh') in stead of an 8-bit general purpose register. This yields instructions that are documented on the R800 and undocumented on the Z80. Similarly 'in f,(c)' is documented on the R800 and undocumented on the Z80.

The assembler also supports the following undocumented Z80-instructions, that have not been adopted in the R800 instruction set:

'out (c),0'

Sends zero to the port pointed to by register c.

'sli M'

Equivalent to 'M = (M<<1)+1', the operand M can be any operand that is valid for 'sla'. One can use 'sll' as a synonym for 'sli'.

'OP (ix+D), R'

This is equivalent to

ld R, (ix+D)

OPC R

ld (ix+D), R

The operation 'OPC' may be any of 'res B,', 'set B,', 'rl', 'rlc', 'rr', 'rrc', 'sla', 'sli', 'sra' and 'srl', and the register 'R' may be any of 'a', 'b', 'c', 'd', 'e', 'h' and 'l'.

'OPC (iy+D), R'

As above, but with 'iy' instead of 'ix'.

The web site at <<http://www.z80.info>> is a good starting place to find more information on programming the Z80.



File: as.info, Node: Z8000-Dependent, Prev: Z80-Dependent, Up: Machine Dependencies

9.55 Z8000 Dependent Features

=====

The Z8000 as supports both members of the Z8000 family: the unsegmented Z8002, with 16 bit addresses, and the segmented Z8001 with 24 bit addresses.

When the assembler is in unsegmented mode (specified with the 'unsegm' directive), an address takes up one word (16 bit) sized register. When the assembler is in segmented mode (specified with the

'segm' directive), a 24-bit address takes up a long (32 bit) register.
 *Note Assembler Directives for the Z8000: Z8000 Directives, for a list of other Z8000 specific assembler directives.

* Menu:

* Z8000 Options::	Command-line options for the Z8000
* Z8000 Syntax::	Assembler syntax for the Z8000
* Z8000 Directives::	Special directives for the Z8000
* Z8000 Opcodes::	Opcodes



File: as.info, Node: Z8000 Options, Next: Z8000 Syntax, Up: Z8000-Dependent

9.55.1 Options

'-z8001'
 Generate segmented code by default.

'-z8002'
 Generate unsegmented code by default.



File: as.info, Node: Z8000 Syntax, Next: Z8000 Directives, Prev: Z8000 Options, Up: Z8000-Dependent

9.55.2 Syntax

* Menu:

* Z8000-Chars::	Special Characters
* Z8000-Regs::	Register Names
* Z8000-Addressing::	Addressing Modes



File: as.info, Node: Z8000-Chars, Next: Z8000-Regs, Up: Z8000 Syntax

9.55.2.1 Special Characters

.....

'!' is the line comment character.

If a '#' appears as the first character of a line then the whole line is treated as a comment, but in this case the line could also be a logical line number directive (*note Comments::) or a preprocessor control command (*note Preprocessing::).

You can use ';' instead of a newline to separate statements.



File: as.info, Node: Z8000-Regs, Next: Z8000-Addressing, Prev: Z8000-Chars, Up: Z8000 Syntax

9.55.2.2 Register Names

.....

The Z8000 has sixteen 16 bit registers, numbered 0 to 15. You can refer

to different sized groups of registers by register number, with the prefix 'r' for 16 bit registers, 'rr' for 32 bit registers and 'rq' for 64 bit registers. You can also refer to the contents of the first eight (of the sixteen 16 bit registers) by bytes. They are named 'r1N' and 'rhN'.

`_byte registers_`

```
r10 rh0 r11 rh1 r12 rh2 r13 rh3
r14 rh4 r15 rh5 r16 rh6 r17 rh7
```

`_word registers_`

```
r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15
```

`_long word registers_`

```
rr0 rr2 rr4 rr6 rr8 rr10 rr12 rr14
```

`_quad word registers_`

```
rq0 rq4 rq8 rq12
```



File: as.info, Node: Z8000-Addressing, Prev: Z8000-Regs, Up: Z8000 Syntax

9.55.2.3 Addressing Modes

.....

as understands the following addressing modes for the Z8000:

```
'r1N'
'rhN'
'rN'
'rrN'
'rqN'
  Register direct: 8bit, 16bit, 32bit, and 64bit registers.
```

```
'@rN'
'@rrN'
  Indirect register: @rrN in segmented mode, @rN in unsegmented mode.
```

```
'ADDR'
  Direct: the 16 bit or 24 bit address (depending on whether the
  assembler is in segmented or unsegmented mode) of the operand is in
  the instruction.
```

```
'address(rN)'
  Indexed: the 16 or 24 bit address is added to the 16 bit register
  to produce the final address in memory of the operand.
```

```
'rN(#IMM)'
'rrN(#IMM)'
  Base Address: the 16 or 24 bit register is added to the 16 bit sign
  extended immediate displacement to produce the final address in
  memory of the operand.
```

```
'rN(rM)'
'rrN(rM)'
  Base Index: the 16 or 24 bit register rN or rrN is added to the
  sign extended 16 bit index register rM to produce the final address
  in memory of the operand.
```

'#XX'
 Immediate data XX.



File: as.info, Node: Z8000 Directives, Next: Z8000 Opcodes, Prev: Z8000 Syntax,
 Up: Z8000-Dependent

9.55.3 Assembler Directives for the Z8000

The Z8000 port of as includes additional assembler directives, for compatibility with other Z8000 assemblers. These do not begin with '.' (unlike the ordinary as directives).

'segm'
 '.z8001'
 Generate code for the segmented Z8001.

'unsegm'
 '.z8002'
 Generate code for the unsegmented Z8002.

'name'
 Synonym for '.file'

'global'
 Synonym for '.global'

'wval'
 Synonym for '.word'

'lval'
 Synonym for '.long'

'bval'
 Synonym for '.byte'

'sval'
 Assemble a string. 'sval' expects one string literal, delimited by single quotes. It assembles each byte of the string into consecutive addresses. You can use the escape sequence '%XX' (where XX represents a two-digit hexadecimal number) to represent the character whose ASCII value is XX. Use this feature to describe single quote and other characters that may not appear in string literals as themselves. For example, the C statement 'char *a = "he said \"it's 50% off\"";' is represented in Z8000 assembly language (shown with the assembler output in hex at the left) as

```

68652073      sval      'he said %22it%27s 50%25 off%22%00'
61696420
22697427
73203530
25206F66
662200

```

'rsect'
 synonym for '.section'

'block'
synonym for '.space'

'even'
special case of '.align'; aligns output to even byte boundary.



File: as.info, Node: Z8000 Opcodes, Prev: Z8000 Directives, Up: Z8000-Dependent

9.55.4 Opcodes

For detailed information on the Z8000 machine instruction set, see 'Z8000 Technical Manual'.

The following table summarizes the opcodes and their arguments:

rs	16 bit source register
rd	16 bit destination register
rbs	8 bit source register
rbd	8 bit destination register
rrs	32 bit source register
rrd	32 bit destination register
rqs	64 bit source register
rqd	64 bit destination register
addr	16/24 bit address
imm	immediate data

adc rd,rs	clrb addr	cpsir @rd,@rs,rr,cc
adcb rbd,rbs	clrb addr(rd)	cpsirb @rd,@rs,rr,cc
add rd,@rs	clrb rbd	dab rbd
add rd,addr	com @rd	dbjnz rbd,disp7
add rd,addr(rs)	com addr	dec @rd,imm4m1
add rd,imm16	com addr(rd)	dec addr(rd),imm4m1
add rd,rs	com rd	dec addr,imm4m1
addb rbd,@rs	comb @rd	dec rd,imm4m1
addb rbd,addr	comb addr	decb @rd,imm4m1
addb rbd,addr(rs)	comb addr(rd)	decb addr(rd),imm4m1
addb rbd,imm8	comb rbd	decb addr,imm4m1
addb rbd,rbs	comflg flags	decb rbd,imm4m1
addl rrd,@rs	cp @rd,imm16	di i2
addl rrd,addr	cp addr(rd),imm16	div rrd,@rs
addl rrd,addr(rs)	cp addr,imm16	div rrd,addr
addl rrd,imm32	cp rd,@rs	div rrd,addr(rs)
addl rrd,rrs	cp rd,addr	div rrd,imm16
and rd,@rs	cp rd,addr(rs)	div rrd,rs
and rd,addr	cp rd,imm16	divl rqd,@rs
and rd,addr(rs)	cp rd,rs	divl rqd,addr
and rd,imm16	cpb @rd,imm8	divl rqd,addr(rs)
and rd,rs	cpb addr(rd),imm8	divl rqd,imm32
andb rbd,@rs	cpb addr,imm8	divl rqd,rrs
andb rbd,addr	cpb rbd,@rs	djnz rd,disp7
andb rbd,addr(rs)	cpb rbd,addr	ei i2
andb rbd,imm8	cpb rbd,addr(rs)	ex rd,@rs
andb rbd,rbs	cpb rbd,imm8	ex rd,addr
bit @rd,imm4	cpb rbd,rbs	ex rd,addr(rs)
bit addr(rd),imm4	cpd rd,@rs,rr,cc	ex rd,rs
bit addr,imm4	cpdb rbd,@rs,rr,cc	exb rbd,@rs
bit rd,imm4	cpdr rd,@rs,rr,cc	exb rbd,addr

bit rd,rs	cpdrb rbd,@rs,rr,cc	exb rbd,addr(rs)
bitb @rd,imm4	cpi rd,@rs,rr,cc	exb rbd,rbs
bitb addr(rd),imm4	cpib rbd,@rs,rr,cc	ext0e imm8
bitb addr,imm4	cpir rd,@rs,rr,cc	ext0f imm8
bitb rbd,imm4	cpirb rbd,@rs,rr,cc	ext8e imm8
bitb rbd,rs	cpl rrd,@rs	ext8f imm8
bpt	cpl rrd,addr	exts rrd
call @rd	cpl rrd,addr(rs)	extsb rd
call addr	cpl rrd,imm32	extsl rqd
call addr(rd)	cpl rrd,rrs	halt
calr disp12	cpsd @rd,@rs,rr,cc	in rd,@rs
clr @rd	cpsdb @rd,@rs,rr,cc	in rd,imm16
clr addr	cpsdr @rd,@rs,rr,cc	inb rbd,@rs
clr addr(rd)	cpsdrb @rd,@rs,rr,cc	inb rbd,imm16
clr rd	cpsib @rd,@rs,rr,cc	inc @rd,imm4m1
clrb @rd	cpsib @rd,@rs,rr,cc	inc addr(rd),imm4m1
inc addr,imm4m1	ldb rbd,rs(rx)	mult rrd,addr(rs)
inc rd,imm4m1	ldb rd(imm16),rbs	mult rrd,imm16
incb @rd,imm4m1	ldb rd(rx),rbs	mult rrd,rs
incb addr(rd),imm4m1	ldctl ctrl,rs	multl rqd,@rs
incb addr,imm4m1	ldctl rd,ctrl	multl rqd,addr
incb rbd,imm4m1	ldd @rs,@rd,rr	multl rqd,addr(rs)
ind @rd,@rs,ra	lddb @rs,@rd,rr	multl rqd,imm32
indb @rd,@rs,rba	lddr @rs,@rd,rr	multl rqd,rrs
inib @rd,@rs,ra	lddrb @rs,@rd,rr	neg @rd
inibr @rd,@rs,ra	ldi @rd,@rs,rr	neg addr
iret	ldib @rd,@rs,rr	neg addr(rd)
jp cc,@rd	ldir @rd,@rs,rr	neg rd
jp cc,addr	ldirb @rd,@rs,rr	negb @rd
jp cc,addr(rd)	ldk rd,imm4	negb addr
jr cc,disp8	ldl @rd,rrs	negb addr(rd)
ld @rd,imm16	ldl addr(rd),rrs	negb rbd
ld @rd,rs	ldl addr,rrs	nop
ld addr(rd),imm16	ldl rd(imm16),rrs	or rd,@rs
ld addr(rd),rs	ldl rd(rx),rrs	or rd,addr
ld addr,imm16	ldl rrd,@rs	or rd,addr(rs)
ld addr,rs	ldl rrd,addr	or rd,imm16
ld rd(imm16),rs	ldl rrd,addr(rs)	or rd,rs
ld rd(rx),rs	ldl rrd,imm32	orb rbd,@rs
ld rd,@rs	ldl rrd,rrs	orb rbd,addr
ld rd,addr	ldl rrd,rs(imm16)	orb rbd,addr(rs)
ld rd,addr(rs)	ldl rrd,rs(rx)	orb rbd,imm8
ld rd,imm16	ldm @rd,rs,n	orb rbd,rbs
ld rd,rs	ldm addr(rd),rs,n	out @rd,rs
ld rd,rs(imm16)	ldm addr,rs,n	out imm16,rs
ld rd,rs(rx)	ldm rd,@rs,n	outb @rd,rbs
lda rd,addr	ldm rd,addr(rs),n	outb imm16,rbs
lda rd,addr(rs)	ldm rd,addr,n	outd @rd,@rs,ra
lda rd,rs(imm16)	ldps @rs	outdb @rd,@rs,rba
lda rd,rs(rx)	ldps addr	outib @rd,@rs,ra
ldar rd,disp16	ldps addr(rs)	outibr @rd,@rs,ra
ldb @rd,imm8	ldr disp16,rs	pop @rd,@rs
ldb @rd,rbs	ldr rd,disp16	pop addr(rd),@rs
ldb addr(rd),imm8	ldrb disp16,rbs	pop addr,@rs
ldb addr(rd),rbs	ldrb rbd,disp16	pop rd,@rs
ldb addr,imm8	ldr1 disp16,rrs	popl @rd,@rs
ldb addr,rbs	ldr1 rrd,disp16	popl addr(rd),@rs
ldb rbd,@rs	mbit	popl addr,@rs
ldb rbd,addr	mreq rd	popl rrd,@rs

ldb rbd,addr(rs)	mres	push @rd,@rs
ldb rbd,imm8	mset	push @rd,addr
ldb rbd,rbs	mult rrd,@rs	push @rd,addr(rs)
ldb rbd,rs(imm16)	mult rrd,addr	push @rd,imm16
push @rd,rs	set addr,imm4	subl rrd,imm32
pushl @rd,@rs	set rd,imm4	subl rrd,rrs
pushl @rd,addr	set rd,rs	tcc cc,rd
pushl @rd,addr(rs)	setb @rd,imm4	tccb cc,rbd
pushl @rd,rrs	setb addr(rd),imm4	test @rd
res @rd,imm4	setb addr,imm4	test addr
res addr(rd),imm4	setb rbd,imm4	test addr(rd)
res addr,imm4	setb rbd,rs	test rd
res rd,imm4	setflg imm4	testb @rd
res rd,rs	sinb rbd,imm16	testb addr
resb @rd,imm4	sinb rd,imm16	testb addr(rd)
resb addr(rd),imm4	sind @rd,@rs,ra	testb rbd
resb addr,imm4	sindb @rd,@rs,rba	testl @rd
resb rbd,imm4	sinib @rd,@rs,ra	testl addr
resb rbd,rs	sinibr @rd,@rs,ra	testl addr(rd)
resflg imm4	sla rd,imm8	testl rrd
ret cc	slab rbd,imm8	trdb @rd,@rs,rba
r1 rd,imm1or2	slal rrd,imm8	trdrb @rd,@rs,rba
rlb rbd,imm1or2	sll rd,imm8	trib @rd,@rs,rbr
rlc rd,imm1or2	sllb rbd,imm8	trirb @rd,@rs,rbr
rlcb rbd,imm1or2	slll rrd,imm8	trtdrb @ra,@rb,rbr
rldb rbb,rba	sout imm16,rs	trtib @ra,@rb,rr
rr rd,imm1or2	soutb imm16,rbs	trtirb @ra,@rb,rbr
rrb rbd,imm1or2	soutd @rd,@rs,ra	trtrb @ra,@rb,rbr
rrc rd,imm1or2	soutdb @rd,@rs,rba	tset @rd
rrcb rbd,imm1or2	soutib @rd,@rs,ra	tset addr
rrdb rbb,rba	soutibr @rd,@rs,ra	tset addr(rd)
rsvd36	sra rd,imm8	tset rd
rsvd38	srab rbd,imm8	tsetb @rd
rsvd78	sral rrd,imm8	tsetb addr
rsvd7e	srl rd,imm8	tsetb addr(rd)
rsvd9d	srlb rbd,imm8	tsetb rbd
rsvd9f	srl1 rrd,imm8	xor rd,@rs
rsvdb9	sub rd,@rs	xor rd,addr
rsvdbf	sub rd,addr	xor rd,addr(rs)
sbc rd,rs	sub rd,addr(rs)	xor rd,imm16
sbc b rbd,rbs	sub rd,imm16	xor rd,rs
sc imm8	sub rd,rs	xorb rbd,@rs
sda rd,rs	subb rbd,@rs	xorb rbd,addr
sdab rbd,rs	subb rbd,addr	xorb rbd,addr(rs)
sdal rrd,rs	subb rbd,addr(rs)	xorb rbd,imm8
sd1 rd,rs	subb rbd,imm8	xorb rbd,rbs
sd1b rbd,rs	subb rbd,rbs	xorb rbd,rbs
sd11 rrd,rs	subl rrd,@rs	
set @rd,imm4	subl rrd,addr	
set addr(rd),imm4	subl rrd,addr(rs)	



File: as.info, Node: Reporting Bugs, Next: Acknowledgements, Prev: Machine Dependencies, Up: Top

10 Reporting Bugs

Your bug reports play an essential role in making 'as' reliable.

Reporting a bug may help you by bringing a solution to your problem, or it may not. But in any case the principal function of a bug report is to help the entire community by making the next version of 'as' work better. Bug reports are your contribution to the maintenance of 'as'.

In order for a bug report to serve its purpose, you must include the information that enables us to fix the bug.

* Menu:

* Bug Criteria:: Have you found a bug?
* Bug Reporting:: How to report bugs

us

File: as.info, Node: Bug Criteria, Next: Bug Reporting, Up: Reporting Bugs

10.1 Have You Found a Bug?

=====

If you are not sure whether you have found a bug, here are some guidelines:

- * If the assembler gets a fatal signal, for any input whatever, that is a 'as' bug. Reliable assemblers never crash.
- * If 'as' produces an error message for valid input, that is a bug.
- * If 'as' does not produce an error message for invalid input, that is a bug. However, you should note that your idea of "invalid input" might be our idea of "an extension" or "support for traditional practice".
- * If you are an experienced user of assemblers, your suggestions for improvement of 'as' are welcome in any case.

us

File: as.info, Node: Bug Reporting, Prev: Bug Criteria, Up: Reporting Bugs

10.2 How to Report Bugs

=====

A number of companies and individuals offer support for GNU products. If you obtained 'as' from a support organization, we recommend you contact that organization first.

You can find contact information for many support companies and individuals in the file 'etc/SERVICE' in the GNU Emacs distribution.

In any event, we also recommend that you send bug reports for 'as' to <http://www.sourceware.org/bugzilla/>.

The fundamental principle of reporting bugs usefully is this: **report all the facts**. If you are not sure whether to state a fact or leave it out, state it!

Often people omit facts because they think they know what causes the problem and assume that some details do not matter. Thus, you might assume that the name of a symbol you use in an example does not matter.

Well, probably it does not, but one cannot be sure. Perhaps the bug is a stray memory reference which happens to fetch from the location where that name is stored in memory; perhaps, if the name were different, the contents of that location would fool the assembler into doing the right thing despite the bug. Play it safe and give a specific, complete example. That is the easiest thing for you to do, and the most helpful.

Keep in mind that the purpose of a bug report is to enable us to fix the bug if it is new to us. Therefore, always write your bug reports on the assumption that the bug has not been reported previously.

Sometimes people give a few sketchy facts and ask, "Does this ring a bell?" This cannot help us fix a bug, so it is basically useless. We respond by asking for enough details to enable us to investigate. You might as well expedite matters by sending them to begin with.

To enable us to fix the bug, you should include all these things:

- * The version of 'as'. 'as' announces it if you start it with the '--version' argument.

Without this, we will not know whether there is any point in looking for the bug in the current version of 'as'.

- * Any patches you may have applied to the 'as' source.
- * The type of machine you are using, and the operating system name and version number.
- * What compiler (and its version) was used to compile 'as'--e.g. "'gcc-2.7'".
- * The command arguments you gave the assembler to assemble your example and observe the bug. To guarantee you will not omit something important, list them all. A copy of the Makefile (or the output from make) is sufficient.

If we were to try to guess the arguments, we would probably guess wrong and then we might not encounter the bug.

- * A complete input file that will reproduce the bug. If the bug is observed when the assembler is invoked via a compiler, send the assembler source, not the high level language source. Most compilers will produce the assembler source when run with the '-S' option. If you are using 'gcc', use the options '-v --save-temps'; this will save the assembler source in a file with an extension of '.s', and also show you exactly how 'as' is being run.
- * A description of what behavior you observe that you believe is incorrect. For example, "It gets a fatal signal."

Of course, if the bug is that 'as' gets a fatal signal, then we will certainly notice it. But if the bug is incorrect output, we might not notice unless it is glaringly wrong. You might as well not give us a chance to make a mistake.

Even if the problem you experience is a fatal signal, you should still say so explicitly. Suppose something strange is going on, such as, your copy of 'as' is out of sync, or you have encountered

a bug in the C library on your system. (This has happened!) Your copy might crash and ours would not. If you told us to expect a crash, then when ours fails to crash, we would know that the bug was not happening for us. If you had not told us to expect a crash, then we would not be able to draw any conclusion from our observations.

- * If you wish to suggest changes to the 'as' source, send us context diffs, as generated by 'diff' with the '-u', '-c', or '-p' option. Always send diffs from the old file to the new file. If you even discuss something in the 'as' source, refer to it by context, not by line number.

The line numbers in our development sources will not match those in your sources. Your line numbers would convey no useful information to us.

Here are some things that are not necessary:

- * A description of the envelope of the bug.

Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it.

This is often time consuming and not very useful, because the way we will find the bug is by running a single example under the debugger with breakpoints, not by pure deduction from a series of examples. We recommend that you save your time for something else.

Of course, if you can find a simpler example to report instead of the original one, that is a convenience for us. Errors in the output will be easier to spot, running under the debugger will take less time, and so on.

However, simplification is not vital; if you do not want to do this, report the bug anyway and send us the entire test case you used.

- * A patch for the bug.

A patch for the bug does help us if it is a good one. But do not omit the necessary information, such as the test case, on the assumption that a patch is all we need. We might see problems with your patch and decide to fix the problem another way, or we might not understand it at all.

Sometimes with a program as complicated as 'as' it is very hard to construct an example that will make the program follow a certain path through the code. If you do not send us the example, we will not be able to construct one, so we will not be able to verify that the bug is fixed.

And if we cannot understand what bug you are trying to fix, or why your patch should be an improvement, we will not install it. A test case will help us to understand.

- * A guess about what the bug is or what it depends on.

Such guesses are usually wrong. Even we cannot guess right about such things without first using the debugger to find the facts.



File: as.info, Node: Acknowledgements, Next: GNU Free Documentation License, Prev: Reporting Bugs, Up: Top

11 Acknowledgements

If you have contributed to GAS and your name isn't listed here, it is not meant as a slight. We just don't know about it. Send mail to the maintainer, and we'll correct the situation. Currently the maintainer is Nick Clifton (email address 'nickc@redhat.com').

Dean Elsner wrote the original GNU assembler for the VAX.(1)

Jay Fenlason maintained GAS for a while, adding support for GDB-specific debug information and the 68k series machines, most of the preprocessing pass, and extensive changes in 'messages.c', 'input-file.c', 'write.c'.

K. Richard Pixley maintained GAS for a while, adding various enhancements and many bug fixes, including merging support for several processors, breaking GAS up to handle multiple object file format back ends (including heavy rewrite, testing, an integration of the coff and b.out back ends), adding configuration including heavy testing and verification of cross assemblers and file splits and renaming, converted GAS to strictly ANSI C including full prototypes, added support for m680[34]0 and cpu32, did considerable work on i960 including a COFF port (including considerable amounts of reverse engineering), a SPARC opcode file rewrite, DECstation, rs6000, and hp300hpux host ports, updated "know" assertions and made them work, much other reorganization, cleanup, and lint.

Ken Raeburn wrote the high-level BFD interface code to replace most of the code in format-specific I/O modules.

The original VMS support was contributed by David L. Kashtan. Eric Youngdale has done much work with it since.

The Intel 80386 machine description was written by Eliot Dresselhaus.

Minh Tran-Le at IntelliCorp contributed some AIX 386 support.

The Motorola 88k machine description was contributed by Devon Bowen of Buffalo University and Torbjorn Granlund of the Swedish Institute of Computer Science.

Keith Knowles at the Open Software Foundation wrote the original MIPS back end ('tc-mips.c', 'tc-mips.h'), and contributed Rose format support (which hasn't been merged in yet). Ralph Campbell worked with the MIPS code to support a.out format.

Support for the Zilog Z8k and Renesas H8/300 processors (tc-z8k, tc-h8300), and IEEE 695 object file format (obj-ieee), was written by Steve Chamberlain of Cygnus Support. Steve also modified the COFF back end to use BFD for some low-level operations, for use with the H8/300 and AMD 29k targets.

John Gilmore built the AMD 29000 support, added '.include' support, and simplified the configuration of which versions accept which directives. He updated the 68k machine description so that Motorola's opcodes always produced fixed-size instructions (e.g., 'jsr'), while synthetic instructions remained shrinkable ('jbsr'). John fixed many bugs, including true tested cross-compilation support, and one bug in relaxation that took a week and required the proverbial one-bit fix.

Ian Lance Taylor of Cygnus Support merged the Motorola and MIT syntax for the 68k, completed support for some COFF targets (68k, i386 SVR3, and SCO Unix), added support for MIPS ECOFF and ELF targets, wrote the initial RS/6000 and PowerPC assembler, and made a few other minor patches.

Steve Chamberlain made GAS able to generate listings.

Hewlett-Packard contributed support for the HP9000/300.

Jeff Law wrote GAS and BFD support for the native HPPA object format (SOM) along with a fairly extensive HPPA testsuite (for both SOM and ELF object formats). This work was supported by both the Center for Software Science at the University of Utah and Cygnus Support.

Support for ELF format files has been worked on by Mark Eichin of Cygnus Support (original, incomplete implementation for SPARC), Pete Hoogenboom and Jeff Law at the University of Utah (HPPA mainly), Michael Meissner of the Open Software Foundation (i386 mainly), and Ken Raeburn of Cygnus Support (sparc, and some initial 64-bit support).

Linus Vepstas added GAS support for the ESA/390 "IBM 370" architecture.

Richard Henderson rewrote the Alpha assembler. Klaus Kaempf wrote GAS and BFD support for openVMS/Alpha.

Timothy Wall, Michael Hayes, and Greg Smart contributed to the various tic* flavors.

David Heine, Sterling Augustine, Bob Wilson and John Ruttenberg from Tensilica, Inc. added support for Xtensa processors.

Several engineers at Cygnus Support have also provided many small bug fixes and configuration enhancements.

Jon Beniston added support for the Lattice Mico32 architecture.

Many others have contributed large or small bugfixes and enhancements. If you have contributed significant work and are not mentioned on this list, and want to be, let us know. Some of the history has been lost; we are not intentionally leaving anyone out.

----- Footnotes -----

(1) Any more details?



Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright (C) 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.

[US](#)

File: as.info, Node: AS Index, Prev: GNU Free Documentation License, Up: Top

AS Index

[ES](#)[[index](#)[ES](#)]

* Menu:

* \" (doublequote character):	Strings.	(line 43)
* \b (backspace character):	Strings.	(line 15)
* \DDD (octal character code):	Strings.	(line 30)
* \f (formfeed character):	Strings.	(line 18)
* \n (newline character):	Strings.	(line 21)
* \r (carriage return character):	Strings.	(line 24)
* \t (tab):	Strings.	(line 27)
* \XD.. ES . (hex character code):	Strings.	(line 36)
* \\ (\ character):	Strings.	(line 40)
* #:	Comments.	(line 33)
* #APP:	Preprocessing.	(line 26)
* #NO_APP:	Preprocessing.	(line 26)
* \$ in symbol names:	D10V-Chars.	(line 46)
* \$ in symbol names <1>:	D30V-Chars.	(line 70)
* \$ in symbol names <2>:	Meta-Chars.	(line 10)
* \$ in symbol names <3>:	SH-Chars.	(line 15)
* \$ in symbol names <4>:	SH64-Chars.	(line 15)
* \$a:	ARM Mapping Symbols.	(line 9)
* \$acos math builtin, TIC54X:	TIC54X-Builtins.	(line 10)
* \$asin math builtin, TIC54X:	TIC54X-Builtins.	(line 13)
* \$atan math builtin, TIC54X:	TIC54X-Builtins.	(line 16)
* \$atan2 math builtin, TIC54X:	TIC54X-Builtins.	(line 19)
* \$ceil math builtin, TIC54X:	TIC54X-Builtins.	(line 22)
* \$cos math builtin, TIC54X:	TIC54X-Builtins.	(line 28)
* \$cosh math builtin, TIC54X:	TIC54X-Builtins.	(line 25)
* \$cvf math builtin, TIC54X:	TIC54X-Builtins.	(line 31)
* \$cvi math builtin, TIC54X:	TIC54X-Builtins.	(line 34)
* \$d:	AArch64 Mapping Symbols.	(line 12)
* \$d <1>:	ARM Mapping Symbols.	(line 15)
* \$exp math builtin, TIC54X:	TIC54X-Builtins.	(line 37)
* \$fabs math builtin, TIC54X:	TIC54X-Builtins.	(line 40)
* \$firstch subsym builtin, TIC54X:	TIC54X-Macros.	(line 26)
* \$floor math builtin, TIC54X:	TIC54X-Builtins.	(line 43)
* \$fmod math builtin, TIC54X:	TIC54X-Builtins.	(line 47)
* \$int math builtin, TIC54X:	TIC54X-Builtins.	(line 50)
* \$iscons subsym builtin, TIC54X:	TIC54X-Macros.	(line 43)
* \$isdefed subsym builtin, TIC54X:	TIC54X-Macros.	(line 34)
* \$ismember subsym builtin, TIC54X:	TIC54X-Macros.	(line 38)
* \$isname subsym builtin, TIC54X:	TIC54X-Macros.	(line 47)

```

* $isreg subsystem builtin, TIC54X:      TIC54X-Macros.      (line 50)
* $lastch subsystem builtin, TIC54X:     TIC54X-Macros.      (line 30)
* $ldexp math builtin, TIC54X:           TIC54X-Builtins.    (line 53)
* $log math builtin, TIC54X:             TIC54X-Builtins.    (line 59)
* $log10 math builtin, TIC54X:           TIC54X-Builtins.    (line 56)
* $max math builtin, TIC54X:             TIC54X-Builtins.    (line 62)
* $min math builtin, TIC54X:             TIC54X-Builtins.    (line 65)
* $pow math builtin, TIC54X:             TIC54X-Builtins.    (line 68)
* $round math builtin, TIC54X:           TIC54X-Builtins.    (line 71)
* $sgn math builtin, TIC54X:             TIC54X-Builtins.    (line 74)
* $sin math builtin, TIC54X:            TIC54X-Builtins.    (line 77)
* $sinh math builtin, TIC54X:           TIC54X-Builtins.    (line 80)
* $sqrt math builtin, TIC54X:           TIC54X-Builtins.    (line 83)
* $structacc subsystem builtin, TIC54X:  TIC54X-Macros.      (line 57)
* $structsz subsystem builtin, TIC54X:   TIC54X-Macros.      (line 54)
* $symcmp subsystem builtin, TIC54X:     TIC54X-Macros.      (line 23)
* $symlen subsystem builtin, TIC54X:     TIC54X-Macros.      (line 20)
* $t:                                     ARM Mapping Symbols. (line 12)

* $tan math builtin, TIC54X:            TIC54X-Builtins.    (line 86)
* $tanh math builtin, TIC54X:           TIC54X-Builtins.    (line 89)
* $trunc math builtin, TIC54X:          TIC54X-Builtins.    (line 92)
* $x:                                     AArch64 Mapping Symbols. (line 9)

* %gp:                                   RX-Modifiers.        (line 6)
* %gpreg:                                RX-Modifiers.        (line 22)
* %pidreg:                                RX-Modifiers.        (line 25)
* +- option, VAX/VMS:                    VAX-Opts.           (line 71)
* --:                                     Command Line.        (line 10)
* --32 option, i386:                     i386-Options.        (line 8)
* --32 option, x86-64:                  i386-Options.        (line 8)
* --64 option, i386:                    i386-Options.        (line 8)
* --64 option, x86-64:                  i386-Options.        (line 8)
* --absolute-literals:                 Xtensa Options.      (line 39)
* --allow-reg-prefix:                  SH Options.          (line 9)
* --alternate:                          alternate.           (line 6)
* --auto-litpools:                     Xtensa Options.      (line 22)
* --base-size-default-16:              M68K-Opts.          (line 66)
* --base-size-default-32:              M68K-Opts.          (line 66)
* --big:                                SH Options.          (line 9)
* --bitwise-or option, M680x0:          M68K-Opts.          (line 59)
* --compress-debug-sections= option:    Overview.           (line 346)
* --disp-size-default-16:              M68K-Opts.          (line 75)
* --disp-size-default-32:              M68K-Opts.          (line 75)
* --divide option, i386:                i386-Options.        (line 24)
* --dsp:                                SH Options.          (line 9)
* --emulation=crisaout command line option, CRIS: CRIS-Opts. (line 9)
* --emulation=criself command line option, CRIS: CRIS-Opts. (line 9)
* --enforce-aligned-data:              Sparc-Aligned-Data. (line 11)
* --fatal-warnings:                    W.                  (line 16)
* --fdpic:                              SH Options.          (line 31)
* --fix-v4bx command line option, ARM:   ARM Options.         (line 191)
* --fixed-special-register-names command line option, MMIX: MMIX-Opts. (line 8)

* --force-long-branches:                M68HC11-Opts.       (line 81)
* --generate-example:                  M68HC11-Opts.       (line 98)
* --globalize-symbols command line option, MMIX: MMIX-Opts. (line 12)
* --gnu-syntax command line option, MMIX: MMIX-Opts. (line 16)
* --linker-allocated-gregs command line option, MMIX: MMIX-Opts.

```



```
(line 67)
* --listing-cont-lines: listing. (line 34)
* --listing-lhs-width: listing. (line 16)
* --listing-lhs-width2: listing. (line 21)
* --listing-rhs-width: listing. (line 28)
* --little: SH Options. (line 9)
* --longcalls: Xtensa Options. (line 53)
* --march=ARCHITECTURE command line option, CRIS: CRIS-Opts. (line 34)
* --MD: MD. (line 6)
* --mul-bug-abort command line option, CRIS: CRIS-Opts. (line 63)
* --no-absolute-literals: Xtensa Options. (line 39)
* --no-auto-litpools: Xtensa Options. (line 22)
* --no-expand command line option, MMIX: MMIX-Opts. (line 31)
* --no-longcalls: Xtensa Options. (line 53)
* --no-merge-gregs command line option, MMIX: MMIX-Opts. (line 36)
* --no-mul-bug-abort command line option, CRIS: CRIS-Opts. (line 63)
* --no-pad-sections: no-pad-sections. (line 6)
* --no-predefined-syms command line option, MMIX: MMIX-Opts. (line 22)
* --no-pushj-stubs command line option, MMIX: MMIX-Opts. (line 54)
* --no-stubs command line option, MMIX: MMIX-Opts. (line 54)
* --no-target-align: Xtensa Options. (line 46)
* --no-text-section-literals: Xtensa Options. (line 7)
* --no-trampolines: Xtensa Options. (line 74)
* --no-transform: Xtensa Options. (line 62)
* --no-underscore command line option, CRIS: CRIS-Opts. (line 15)
* --no-warn: W. (line 11)
* --pcrel: M68K-Opts. (line 87)
* --pic command line option, CRIS: CRIS-Opts. (line 27)
* --print-insn-syntax: M68HC11-Opts. (line 87)
* --print-insn-syntax <1>: XGATE-Opts. (line 25)
* --print-opcodes: M68HC11-Opts. (line 91)
* --print-opcodes <1>: XGATE-Opts. (line 29)
* --register-prefix-optional option, M680x0: M68K-Opts. (line 46)
* --relax: SH Options. (line 9)
* --relax command line option, MMIX: MMIX-Opts. (line 19)
* --rename-section: Xtensa Options. (line 70)
* --renesas: SH Options. (line 9)
* --sectname-subst: Section. (line 71)
* --short-branches: M68HC11-Opts. (line 67)
* --small: SH Options. (line 9)
* --statistics: statistics. (line 6)
* --strict-direct-mode: M68HC11-Opts. (line 57)
* --target-align: Xtensa Options. (line 46)
* --text-section-literals: Xtensa Options. (line 7)
* --traditional-format: traditional-format. (line 6)
* --trampolines: Xtensa Options. (line 74)
* --transform: Xtensa Options. (line 62)
* --underscore command line option, CRIS: CRIS-Opts. (line 15)
* --warn: W. (line 19)
* --x32 option, i386: i386-Options. (line 8)
* --x32 option, x86-64: i386-Options. (line 8)
* --xgate-ramoffset: M68HC11-Opts. (line 36)
* -1 option, VAX/VMS: VAX-Opts. (line 77)
* -32addr command line option, Alpha: Alpha Options. (line 57)
* -a: a. (line 6)
* -A options, i960: Options-i960. (line 6)
* -ac: a. (line 6)
* -ad: a. (line 6)
* -ag: a. (line 6)
```

* -ah:	a.	(line 6)
* -al:	a.	(line 6)
* -Aleon:	Sparc-Opts.	(line 25)
* -an:	a.	(line 6)
* -as:	a.	(line 6)
* -Asparc:	Sparc-Opts.	(line 25)
* -Asparcfmaf:	Sparc-Opts.	(line 25)
* -Asparcima:	Sparc-Opts.	(line 25)
* -Asparclet:	Sparc-Opts.	(line 25)
* -Asparclite:	Sparc-Opts.	(line 25)
* -Asparcvis:	Sparc-Opts.	(line 25)
* -Asparcvis2:	Sparc-Opts.	(line 25)
* -Asparcvis3:	Sparc-Opts.	(line 25)
* -Asparcvis3r:	Sparc-Opts.	(line 25)
* -Av6:	Sparc-Opts.	(line 25)
* -Av7:	Sparc-Opts.	(line 25)
* -Av8:	Sparc-Opts.	(line 25)
* -Av9:	Sparc-Opts.	(line 25)
* -Av9a:	Sparc-Opts.	(line 25)
* -Av9b:	Sparc-Opts.	(line 25)
* -Av9c:	Sparc-Opts.	(line 25)
* -Av9d:	Sparc-Opts.	(line 25)
* -Av9e:	Sparc-Opts.	(line 25)
* -Av9m:	Sparc-Opts.	(line 25)
* -Av9v:	Sparc-Opts.	(line 25)
* -b option, i960:	Options-i960.	(line 22)
* -big option, M32R:	M32R-Opts.	(line 35)
* -D:	D.	(line 6)
* -D, ignored on VAX:	VAX-Opts.	(line 11)
* -d, VAX option:	VAX-Opts.	(line 16)
* -eabi= command line option, ARM:	ARM Options.	(line 167)
* -EB command line option, AArch64:	AArch64 Options.	(line 6)
* -EB command line option, ARC:	ARC Options.	(line 84)
* -EB command line option, ARM:	ARM Options.	(line 172)
* -EB option (MIPS):	MIPS Options.	(line 13)
* -EB option, M32R:	M32R-Opts.	(line 39)
* -EB option, TILE-Gx:	TILE-Gx Options.	(line 11)
* -EL command line option, AArch64:	AArch64 Options.	(line 10)
* -EL command line option, ARC:	ARC Options.	(line 88)
* -EL command line option, ARM:	ARM Options.	(line 183)
* -EL option (MIPS):	MIPS Options.	(line 13)
* -EL option, M32R:	M32R-Opts.	(line 32)
* -EL option, TILE-Gx:	TILE-Gx Options.	(line 11)
* -f:	f.	(line 6)
* -F command line option, Alpha:	Alpha Options.	(line 57)
* -fno-pic option, RISC-V:	RISC-V-Opts.	(line 11)
* -fpic option, RISC-V:	RISC-V-Opts.	(line 8)
* -g command line option, Alpha:	Alpha Options.	(line 47)
* -G command line option, Alpha:	Alpha Options.	(line 53)
* -G option (MIPS):	MIPS Options.	(line 8)
* -h option, VAX/VMS:	VAX-Opts.	(line 45)
* -H option, VAX/VMS:	VAX-Opts.	(line 81)
* -I PATH:	I.	(line 6)
* -ignore-parallel-conflicts option, M32RX:	M32R-Opts.	(line 87)
* -Ip option, M32RX:	M32R-Opts.	(line 97)
* -J, ignored on VAX:	VAX-Opts.	(line 27)
* -K:	K.	(line 6)
* -k command line option, ARM:	ARM Options.	(line 187)
* -KPIC option, M32R:	M32R-Opts.	(line 42)

```

* -KPIC option, MIPS: MIPS Options. (line 21)
* -L: L. (line 6)
* -l option, M680x0: M68K-Opts. (line 34)
* -little option, M32R: M32R-Opts. (line 27)
* -M: M. (line 6)
* -m11/03: PDP-11-Options. (line 140)
* -m11/04: PDP-11-Options. (line 143)
* -m11/05: PDP-11-Options. (line 146)
* -m11/10: PDP-11-Options. (line 146)
* -m11/15: PDP-11-Options. (line 149)
* -m11/20: PDP-11-Options. (line 149)
* -m11/21: PDP-11-Options. (line 152)
* -m11/23: PDP-11-Options. (line 155)
* -m11/24: PDP-11-Options. (line 155)
* -m11/34: PDP-11-Options. (line 158)
* -m11/34a: PDP-11-Options. (line 161)
* -m11/35: PDP-11-Options. (line 164)
* -m11/40: PDP-11-Options. (line 164)
* -m11/44: PDP-11-Options. (line 167)
* -m11/45: PDP-11-Options. (line 170)
* -m11/50: PDP-11-Options. (line 170)
* -m11/53: PDP-11-Options. (line 173)
* -m11/55: PDP-11-Options. (line 170)
* -m11/60: PDP-11-Options. (line 176)
* -m11/70: PDP-11-Options. (line 170)
* -m11/73: PDP-11-Options. (line 173)
* -m11/83: PDP-11-Options. (line 173)
* -m11/84: PDP-11-Options. (line 173)
* -m11/93: PDP-11-Options. (line 173)
* -m11/94: PDP-11-Options. (line 173)
* -m16c option, M16C: M32C-Opts. (line 12)
* -m31 option, s390: s390 Options. (line 8)
* -m32 option, TILE-Gx: TILE-Gx Options. (line 8)
* -m32bit-doubles: RX-Opts. (line 9)
* -m32c option, M32C: M32C-Opts. (line 9)
* -m32r option, M32R: M32R-Opts. (line 21)
* -m32rx option, M32R2: M32R-Opts. (line 17)
* -m32rx option, M32RX: M32R-Opts. (line 9)
* -m4byte-align command line option, V850: V850 Options. (line 90)
* -m64 option, s390: s390 Options. (line 8)
* -m64 option, TILE-Gx: TILE-Gx Options. (line 8)
* -m64bit-doubles: RX-Opts. (line 15)
* -m68000 and related options: M68K-Opts. (line 99)
* -m68hc11: M68HC11-Opts. (line 9)
* -m68hc12: M68HC11-Opts. (line 14)
* -m68hcs12: M68HC11-Opts. (line 21)
* -m8byte-align command line option, V850: V850 Options. (line 86)
* -mabi= command line option, AArch64: AArch64 Options. (line 14)
* -mabi=ABI option, RISC-V: RISC-V-Opts. (line 18)
* -madd-bnd-prefix option, i386: i386-Options. (line 134)
* -madd-bnd-prefix option, x86-64: i386-Options. (line 134)
* -mall: PDP-11-Options. (line 26)
* -mall-enabled command line option, LM32: LM32 Options. (line 30)
* -mall-extensions: PDP-11-Options. (line 26)
* -mall-opcodes command line option, AVR: AVR Options. (line 108)
* -mamd64 option, x86-64: i386-Options. (line 190)
* -mapcs-26 command line option, ARM: ARM Options. (line 139)
* -mapcs-32 command line option, ARM: ARM Options. (line 139)
* -mapcs-float command line option, ARM: ARM Options. (line 153)

```

```

* -mapcs-reentrant command line option, ARM: ARM Options. (line 158)
* -march= command line option, AArch64: AArch64 Options. (line 39)
* -march= command line option, ARM: ARM Options. (line 73)
* -march= command line option, M680x0: M68K-Opts. (line 8)
* -march= command line option, TIC6X: TIC6X Options. (line 6)
* -march= option, i386: i386-Options. (line 31)
* -march= option, s390: s390 Options. (line 25)
* -march= option, x86-64: i386-Options. (line 31)
* -march=ISA option, RISC-V: RISC-V-Opts. (line 14)
* -matpcs command line option, ARM: ARM Options. (line 145)
* -mavxscalar= option, i386: i386-Options. (line 92)
* -mavxscalar= option, x86-64: i386-Options. (line 92)
* -mbarrel-shift-enabled command line option, LM32: LM32 Options. (line 12)
* -mbig-endian: RX-Opts. (line 20)
* -mbig-obj option, x86-64: i386-Options. (line 148)
* -mbreak-enabled command line option, LM32: LM32 Options. (line 27)
* -mccs command line option, ARM: ARM Options. (line 200)
* -mcis: PDP-11-Options. (line 32)
* -mcode-density command line option, ARC: ARC Options. (line 93)
* -mconstant-gp command line option, IA-64: IA-64 Options. (line 6)
* -mCPU command line option, Alpha: Alpha Options. (line 6)
* -mcpu option, cpu: TIC54X-Opts. (line 15)
* -mcpu=: RX-Opts. (line 75)
* -mcpu= command line option, AArch64: AArch64 Options. (line 19)
* -mcpu= command line option, ARM: ARM Options. (line 6)
* -mcpu= command line option, Blackfin: Blackfin Options. (line 6)
* -mcpu= command line option, M680x0: M68K-Opts. (line 14)
* -mcpu=CPU command line option, ARC: ARC Options. (line 10)
* -mcs: PDP-11-Options. (line 43)
* -mdcache-enabled command line option, LM32: LM32 Options. (line 24)
* -mdebug command line option, Alpha: Alpha Options. (line 25)
* -mdivide-enabled command line option, LM32: LM32 Options. (line 9)
* -mdpfp command line option, ARC: ARC Options. (line 108)
* -mdsbt command line option, TIC6X: TIC6X Options. (line 13)
* -me option, stderr redirect: TIC54X-Opts. (line 20)
* -meis: PDP-11-Options. (line 46)
* -mepiphany command line option, Epiphany: Epiphany Options. (line 9)
* -mepiphany16 command line option, Epiphany: Epiphany Options. (line 13)
* -merrors-to-file option, stderr redirect: TIC54X-Opts. (line 20)
* -mesa option, s390: s390 Options. (line 17)
* -mevexlig= option, i386: i386-Options. (line 100)
* -mevexlig= option, x86-64: i386-Options. (line 100)
* -mevexcig= option, i386: i386-Options. (line 180)
* -mevexcig= option, x86-64: i386-Options. (line 180)
* -mevexwig= option, i386: i386-Options. (line 110)
* -mevexwig= option, x86-64: i386-Options. (line 110)
* -mf option, far-mode: TIC54X-Opts. (line 8)
* -mf11: PDP-11-Options. (line 122)
* -mfar-mode option, far-mode: TIC54X-Opts. (line 8)
* -mfdpic command line option, Blackfin: Blackfin Options. (line 19)
* -mfence-as-lock-add= option, i386: i386-Options. (line 161)
* -mfence-as-lock-add= option, x86-64: i386-Options. (line 161)
* -mfis: PDP-11-Options. (line 51)
* -mfloat-abi= command line option, ARM: ARM Options. (line 162)
* -mfp-11: PDP-11-Options. (line 56)
* -mfpp: PDP-11-Options. (line 56)

```

```

* -mfpu: PDP-11-Options. (line 56)
* -mfpu= command line option, ARM: ARM Options. (line 90)
* -mfpuda command line option, ARC: ARC Options. (line 111)
* -mgcc-abi: RX-Opts. (line 63)
* -mgcc-abi command line option, V850: V850 Options. (line 79)
* -mhard-float command line option, V850: V850 Options. (line 101)
* -micache-enabled command line option, LM32: LM32 Options. (line 21)
* -mimplicit-it command line option, ARM: ARM Options. (line 123)
* -mint-register: RX-Opts. (line 57)
* -mintel64 option, x86-64: i386-Options. (line 190)
* -mip2022 option, IP2K: IP2K-Opts. (line 14)
* -mip2022ext option, IP2022: IP2K-Opts. (line 9)
* -mj11: PDP-11-Options. (line 126)
* -mka11: PDP-11-Options. (line 92)
* -mkb11: PDP-11-Options. (line 95)
* -mkd11a: PDP-11-Options. (line 98)
* -mkd11b: PDP-11-Options. (line 101)
* -mkd11d: PDP-11-Options. (line 104)
* -mkd11e: PDP-11-Options. (line 107)
* -mkd11f: PDP-11-Options. (line 110)
* -mkd11h: PDP-11-Options. (line 110)
* -mkd11k: PDP-11-Options. (line 114)
* -mkd11q: PDP-11-Options. (line 110)
* -mkd11z: PDP-11-Options. (line 118)
* -mkev11: PDP-11-Options. (line 51)
* -mkev11 <1>: PDP-11-Options. (line 51)
* -mlimited-eis: PDP-11-Options. (line 64)
* -mlink-relax command line option, AVR: AVR Options. (line 120)
* -mlittle-endian: RX-Opts. (line 26)
* -mlong: M68HC11-Opts. (line 45)
* -mlong <1>: XGATE-Opts. (line 13)
* -mlong-double: M68HC11-Opts. (line 53)
* -mlong-double <1>: XGATE-Opts. (line 21)
* -mm9s12x: M68HC11-Opts. (line 27)
* -mm9s12xg: M68HC11-Opts. (line 32)
* -mmcu= command line option, AVR: AVR Options. (line 6)
* -mmfpt: PDP-11-Options. (line 70)
* -mmicrocode: PDP-11-Options. (line 83)
* -mmnemonic= option, i386: i386-Options. (line 117)
* -mmnemonic= option, x86-64: i386-Options. (line 117)
* -mmultiply-enabled command line option, LM32: LM32 Options. (line 6)
* -mmutiproc: PDP-11-Options. (line 73)
* -mmxps: PDP-11-Options. (line 77)
* -mnaked-reg option, i386: i386-Options. (line 129)
* -mnaked-reg option, x86-64: i386-Options. (line 129)
* -mnan= command line option, MIPS: MIPS Options. (line 379)
* -mno-allow-string-insns: RX-Opts. (line 82)
* -mno-cis: PDP-11-Options. (line 32)
* -mno-csm: PDP-11-Options. (line 43)
* -mno-dsbt command line option, TIC6X: TIC6X Options. (line 13)
* -mno-eis: PDP-11-Options. (line 46)
* -mno-extensions: PDP-11-Options. (line 29)
* -mno-fdpic command line option, Blackfin: Blackfin Options. (line 22)
* -mno-fis: PDP-11-Options. (line 51)
* -mno-fp-11: PDP-11-Options. (line 56)
* -mno-fpp: PDP-11-Options. (line 56)
* -mno-fpu: PDP-11-Options. (line 56)

```

```

* -mno-kev11: PDP-11-Options. (line 51)
* -mno-limited-eis: PDP-11-Options. (line 64)
* -mno-link-relax command line option, AVR: AVR Options. (line 124)
* -mno-mfpt: PDP-11-Options. (line 70)
* -mno-microcode: PDP-11-Options. (line 83)
* -mno-mutiproc: PDP-11-Options. (line 73)
* -mno-mxps: PDP-11-Options. (line 77)
* -mno-pic: PDP-11-Options. (line 11)
* -mno-pic command line option, TIC6X: TIC6X Options. (line 36)
* -mno-regnames option, s390: s390 Options. (line 50)
* -mno-skip-bug command line option, AVR: AVR Options. (line 111)
* -mno-spl: PDP-11-Options. (line 80)
* -mno-sym32: MIPS Options. (line 288)
* -mno-verbose-error command line option, AArch64: AArch64 Options.
(line 59)
* -mno-wrap command line option, AVR: AVR Options. (line 114)
* -mnopic command line option, Blackfin: Blackfin Options. (line 22)
* -mnps400 command line option, ARC: ARC Options. (line 102)
* -momit-lock-prefix= option, i386: i386-Options. (line 152)
* -momit-lock-prefix= option, x86-64: i386-Options. (line 152)
* -mpic: PDP-11-Options. (line 11)
* -mpic command line option, TIC6X: TIC6X Options. (line 36)
* -mpid: RX-Opts. (line 50)
* -mpid= command line option, TIC6X: TIC6X Options. (line 23)
* -mregnames option, s390: s390 Options. (line 47)
* -mrelax command line option, ARC: ARC Options. (line 97)
* -mrelax command line option, V850: V850 Options. (line 72)
* -mrelax-relocations= option, i386: i386-Options. (line 170)
* -mrelax-relocations= option, x86-64: i386-Options. (line 170)
* -mrh850-abi command line option, V850: V850 Options. (line 82)
* -mrmw command line option, AVR: AVR Options. (line 117)
* -mrx-abi: RX-Opts. (line 69)
* -mshared option, i386: i386-Options. (line 139)
* -mshared option, x86-64: i386-Options. (line 139)
* -mshort: M68HC11-Opts. (line 40)
* -mshort <1>: XGATE-Opts. (line 8)
* -mshort-double: M68HC11-Opts. (line 49)
* -mshort-double <1>: XGATE-Opts. (line 17)
* -msign-extend-enabled command line option, LM32: LM32 Options.
(line 15)
* -msmall-data-limit: RX-Opts. (line 42)
* -msoft-float command line option, V850: V850 Options. (line 95)
* -mspfp command line option, ARC: ARC Options. (line 105)
* -mspl: PDP-11-Options. (line 80)
* -msse-check= option, i386: i386-Options. (line 82)
* -msse-check= option, x86-64: i386-Options. (line 82)
* -msse2avx option, i386: i386-Options. (line 78)
* -msse2avx option, x86-64: i386-Options. (line 78)
* -msym32: MIPS Options. (line 288)
* -msyntax= option, i386: i386-Options. (line 123)
* -msyntax= option, x86-64: i386-Options. (line 123)
* -mt11: PDP-11-Options. (line 130)
* -mthumb command line option, ARM: ARM Options. (line 114)
* -mthumb-interwork command line option, ARM: ARM Options. (line 119)
* -mtune= option, i386: i386-Options. (line 70)
* -mtune= option, x86-64: i386-Options. (line 70)
* -mtune=ARCH command line option, Visium: Visium Options. (line 8)
* -muse-conventional-section-names: RX-Opts. (line 33)
* -muse-renesas-section-names: RX-Opts. (line 37)

```

```

* -muser-enabled command line option, LM32: LM32 Options. (line 18)
* -mv850 command line option, V850: V850 Options. (line 23)
* -mv850any command line option, V850: V850 Options. (line 41)
* -mv850e command line option, V850: V850 Options. (line 29)
* -mv850e1 command line option, V850: V850 Options. (line 35)
* -mv850e2 command line option, V850: V850 Options. (line 51)
* -mv850e2v3 command line option, V850: V850 Options. (line 57)
* -mv850e2v4 command line option, V850: V850 Options. (line 63)
* -mv850e3v5 command line option, V850: V850 Options. (line 66)
* -mverbose-error command line option, AArch64: AArch64 Options.
 (line 55)
* -mvxworks-pic option, MIPS: MIPS Options. (line 26)
* -mwarn-areg-zero option, s390: s390 Options. (line 53)
* -mwarn-deprecated command line option, ARM: ARM Options. (line 195)
* -mwarn-syms command line option, ARM: ARM Options. (line 203)
* -mzarch option, s390: s390 Options. (line 17)
* -m[no-]68851 command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]68881 command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]div command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]emac command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]float command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]mac command line option, M680x0: M68K-Opts. (line 21)
* -m[no-]usp command line option, M680x0: M68K-Opts. (line 21)
* -N command line option, CRIS: CRIS-Opts. (line 59)
* -nIp option, M32RX: M32R-Opts. (line 101)
* -no-bitinst, M32R2: M32R-Opts. (line 54)
* -no-ignore-parallel-conflicts option, M32RX: M32R-Opts. (line 93)
* -no-mdebug command line option, Alpha: Alpha Options. (line 25)
* -no-parallel option, M32RX: M32R-Opts. (line 51)
* -no-relax option, i960: Options-i960. (line 66)
* -no-warn-explicit-parallel-conflicts option, M32RX: M32R-Opts.
 (line 79)
* -no-warn-unmatched-high option, M32R: M32R-Opts. (line 111)
* -nocpp ignored (MIPS): MIPS Options. (line 291)
* -noreplace command line option, Alpha: Alpha Options. (line 40)
* -o: o. (line 6)
* -O option, M32RX: M32R-Opts. (line 59)
* -parallel option, M32RX: M32R-Opts. (line 46)
* -R: R. (line 6)
* -r800 command line option, Z80: Z80 Options. (line 35)
* -relax command line option, Alpha: Alpha Options. (line 32)
* -replace command line option, Alpha: Alpha Options. (line 40)
* -S, ignored on VAX: VAX-Opts. (line 11)
* -T, ignored on VAX: VAX-Opts. (line 11)
* -t, ignored on VAX: VAX-Opts. (line 36)
* -v: v. (line 6)
* -V, redundant on VAX: VAX-Opts. (line 22)
* -version: v. (line 6)
* -W: W. (line 11)
* -warn-explicit-parallel-conflicts option, M32RX: M32R-Opts.
 (line 65)
* -warn-unmatched-high option, M32R: M32R-Opts. (line 105)
* -Wnp option, M32RX: M32R-Opts. (line 83)
* -Wnuh option, M32RX: M32R-Opts. (line 117)
* -Wp option, M32RX: M32R-Opts. (line 75)
* -wsigned_overflow command line option, V850: V850 Options. (line 9)
* -Wuh option, M32RX: M32R-Opts. (line 114)
* -wunsigned_overflow command line option, V850: V850 Options.
 (line 16)

```

```

* -x command line option, MMIX: MMIX-Opts. (line 44)
* -z80 command line option, Z80: Z80 Options. (line 8)
* -z8001 command line option, Z8000: Z8000 Options. (line 6)
* -z8002 command line option, Z8000: Z8000 Options. (line 9)
* . (symbol): Dot. (line 6)
* .2byte directive, ARM: ARM Directives. (line 6)
* .4byte directive, ARM: ARM Directives. (line 6)
* .8byte directive, ARM: ARM Directives. (line 6)
* .align directive, ARM: ARM Directives. (line 11)
* .align directive, TILE-Gx: TILE-Gx Directives. (line 6)
* .align directive, TILEPro: TILEPro Directives. (line 6)
* .allow_suspicious_bundles directive, TILE-Gx: TILE-Gx Directives.
(line 10)
* .allow_suspicious_bundles directive, TILEPro: TILEPro Directives.
(line 10)
* .arch directive, AArch64: AArch64 Directives. (line 6)
* .arch directive, ARM: ARM Directives. (line 18)
* .arch directive, TIC6X: TIC6X Directives. (line 10)
* .arch_extension directive, AArch64: AArch64 Directives. (line 13)
* .arch_extension directive, ARM: ARM Directives. (line 25)
* .arm directive, ARM: ARM Directives. (line 33)
* .big directive, M32RX: M32R-Directives. (line 88)
* .bss directive, AArch64: AArch64 Directives. (line 21)
* .bss directive, ARM: ARM Directives. (line 36)
* .c6xabi_attribute directive, TIC6X: TIC6X Directives. (line 20)
* .cantunwind directive, ARM: ARM Directives. (line 39)
* .cantunwind directive, TIC6X: TIC6X Directives. (line 13)
* .code directive, ARM: ARM Directives. (line 43)
* .cpu directive, AArch64: AArch64 Directives. (line 24)
* .cpu directive, ARM: ARM Directives. (line 47)
* .dn and .qn directives, ARM: ARM Directives. (line 54)
* .dword directive, AArch64: AArch64 Directives. (line 28)
* .eabi_attribute directive, ARM: ARM Directives. (line 78)
* .eh_type directive, TIC6X: TIC6X Directives. (line 31)
* .endp directive, TIC6X: TIC6X Directives. (line 34)
* .even directive, AArch64: AArch64 Directives. (line 31)
* .even directive, ARM: ARM Directives. (line 106)
* .extend directive, ARM: ARM Directives. (line 109)
* .fnend directive, ARM: ARM Directives. (line 115)
* .fnstart directive, ARM: ARM Directives. (line 123)
* .force_thumb directive, ARM: ARM Directives. (line 126)
* .fpu directive, ARM: ARM Directives. (line 130)
* .global: MIPS insn. (line 12)
* .gnu_attribute 4, N directive, MIPS: MIPS FP ABI History.
(line 6)
* .gnu_attribute Tag_GNU_MIPS_ABI_FP, N directive, MIPS: MIPS FP ABI History.
(line 6)
* .handlerdata directive, ARM: ARM Directives. (line 134)
* .handlerdata directive, TIC6X: TIC6X Directives. (line 39)
* .insn: MIPS insn. (line 6)
* .insn directive, s390: s390 Directives. (line 11)
* .inst directive, AArch64: AArch64 Directives. (line 35)
* .inst directive, ARM: ARM Directives. (line 143)
* .ldouble directive, ARM: ARM Directives. (line 109)
* .little directive, M32RX: M32R-Directives. (line 82)
* .long directive, s390: s390 Directives. (line 16)
* .ltorg directive, AArch64: AArch64 Directives. (line 39)
* .ltorg directive, ARM: ARM Directives. (line 153)
* .ltorg directive, s390: s390 Directives. (line 79)

```



```

* .m32r directive, M32R:      M32R-Directives.      (line 66)
* .m32r2 directive, M32R2:    M32R-Directives.      (line 77)
* .m32rx directive, M32RX:    M32R-Directives.      (line 72)
* .machine directive, s390:    s390 Directives.      (line 84)
* .machinemode directive, s390: s390 Directives.      (line 101)
* .module:                     MIPS assembly options.
                                (line 6)
* .module fp=NN directive, MIPS: MIPS FP ABI Selection.
                                (line 6)
* .movsp directive, ARM:       ARM Directives.        (line 167)
* .nan directive, MIPS:        MIPS NaN Encodings.    (line 6)
* .nocmp directive, TIC6X:     TIC6X Directives.      (line 47)
* .no_pointers directive, XStormy16: XStormy16 Directives.
                                (line 14)
* .o:                           Object.                  (line 6)
* .object_arch directive, ARM:  ARM Directives.        (line 172)
* .packed directive, ARM:      ARM Directives.        (line 178)
* .pad directive, ARM:         ARM Directives.        (line 183)
* .param on HPPA:             HPPA Directives.      (line 19)
* .personality directive, ARM:  ARM Directives.        (line 188)
* .personality directive, TIC6X: TIC6X Directives.      (line 55)
* .personalityindex directive, ARM: ARM Directives.        (line 191)
* .personalityindex directive, TIC6X: TIC6X Directives.      (line 51)
* .pool directive, AArch64:    AArch64 Directives.   (line 53)
* .pool directive, ARM:        ARM Directives.        (line 195)
* .quad directive, s390:       s390 Directives.      (line 16)
* .req directive, AArch64:     AArch64 Directives.   (line 56)
* .req directive, ARM:         ARM Directives.        (line 198)
* .require_canonical_reg_names directive, TILE-Gx: TILE-Gx Directives.
                                (line 19)
* .require_canonical_reg_names directive, TILEPro: TILEPro Directives.
                                (line 19)
* .save directive, ARM:        ARM Directives.        (line 203)
* .scomm directive, TIC6X:     TIC6X Directives.      (line 58)
* .secret32 directive, ARM:    ARM Directives.        (line 241)
* .set arch=CPU:               MIPS ISA.                (line 18)
* .set at:                     MIPS Macros.             (line 41)
* .set at=REG:                 MIPS Macros.             (line 35)
* .set autoextend:             MIPS autoextend.        (line 6)
* .set doublefloat:           MIPS Floating-Point.
                                (line 12)
* .set dsp:                    MIPS ASE Instruction Generation Overrides.
                                (line 21)
* .set dspr2:                  MIPS ASE Instruction Generation Overrides.
                                (line 26)
* .set dspr3:                  MIPS ASE Instruction Generation Overrides.
                                (line 31)
* .set hardfloat:             MIPS Floating-Point.
                                (line 6)
* .set insn32:                 MIPS assembly options.
                                (line 18)
* .set macro:                  MIPS Macros.             (line 30)
* .set mcu:                    MIPS ASE Instruction Generation Overrides.
                                (line 42)
* .set mdmx:                   MIPS ASE Instruction Generation Overrides.
                                (line 16)
* .set mips3d:                 MIPS ASE Instruction Generation Overrides.
                                (line 6)
* .set mipsN:                  MIPS ISA.                (line 6)

```

```

* .set msa: MIPS ASE Instruction Generation Overrides.
              (line 47)
* .set mt: MIPS ASE Instruction Generation Overrides.
              (line 37)
* .set noat: MIPS Macros. (line 41)
* .set noautoextend: MIPS autoextend. (line 6)
* .set nodsp: MIPS ASE Instruction Generation Overrides.
              (line 21)
* .set nodspr2: MIPS ASE Instruction Generation Overrides.
              (line 26)
* .set nodspr3: MIPS ASE Instruction Generation Overrides.
              (line 31)
* .set noinsn32: MIPS assembly options.
              (line 18)
* .set nomacro: MIPS Macros. (line 30)
* .set nomcu: MIPS ASE Instruction Generation Overrides.
              (line 42)
* .set nomdmx: MIPS ASE Instruction Generation Overrides.
              (line 16)
* .set nomips3d: MIPS ASE Instruction Generation Overrides.
              (line 6)
* .set nomsa: MIPS ASE Instruction Generation Overrides.
              (line 47)
* .set nomt: MIPS ASE Instruction Generation Overrides.
              (line 37)
* .set nosmartmips: MIPS ASE Instruction Generation Overrides.
              (line 11)
* .set nosym32: MIPS Symbol Sizes. (line 6)
* .set novirt: MIPS ASE Instruction Generation Overrides.
              (line 52)
* .set noxpa: MIPS ASE Instruction Generation Overrides.
              (line 57)
* .set pop: MIPS Option Stack. (line 6)
* .set push: MIPS Option Stack. (line 6)
* .set singlefloat: MIPS Floating-Point.
              (line 12)
* .set smartmips: MIPS ASE Instruction Generation Overrides.
              (line 11)
* .set softfloat: MIPS Floating-Point.
              (line 6)
* .set sym32: MIPS Symbol Sizes. (line 6)
* .set virt: MIPS ASE Instruction Generation Overrides.
              (line 52)
* .set xpa: MIPS ASE Instruction Generation Overrides.
              (line 57)
* .setfp directive, ARM: ARM Directives. (line 227)
* .short directive, s390: s390 Directives. (line 16)
* .syntax directive, ARM: ARM Directives. (line 246)
* .thumb directive, ARM: ARM Directives. (line 250)
* .thumb_func directive, ARM: ARM Directives. (line 253)
* .thumb_set directive, ARM: ARM Directives. (line 264)
* .tlsdescadd directive, AArch64: AArch64 Directives. (line 61)
* .tlsdesccall directive, AArch64: AArch64 Directives. (line 64)
* .tlsdescldr directive, AArch64: AArch64 Directives. (line 67)
* .tlsdescseq directive, ARM: ARM Directives. (line 271)
* .unreq directive, AArch64: AArch64 Directives. (line 70)
* .unreq directive, ARM: ARM Directives. (line 276)
* .unwind_raw directive, ARM: ARM Directives. (line 287)
* .v850 directive, V850: V850 Directives. (line 14)

```

```

* .v850e directive, V850: V850 Directives. (line 20)
* .v850e1 directive, V850: V850 Directives. (line 26)
* .v850e2 directive, V850: V850 Directives. (line 32)
* .v850e2v3 directive, V850: V850 Directives. (line 38)
* .v850e2v4 directive, V850: V850 Directives. (line 44)
* .v850e3v5 directive, V850: V850 Directives. (line 50)
* .vsave directive, ARM: ARM Directives. (line 294)
* .xword directive, AArch64: AArch64 Directives. (line 81)
* .z8001: Z8000 Directives. (line 11)
* .z8002: Z8000 Directives. (line 15)
* 16-bit code, i386: i386-16bit. (line 6)
* 16bit_pointers directive, XStormy16: XStormy16 Directives.
                                         (line 6)
* 16byte directive, Nios II: Nios II Directives. (line 28)
* 2byte directive, Nios II: Nios II Directives. (line 19)
* 32bit_pointers directive, XStormy16: XStormy16 Directives.
                                         (line 10)
* 3DNow!, i386: i386-SIMD. (line 6)
* 3DNow!, x86-64: i386-SIMD. (line 6)
* 430 support: MSP430-Dependent. (line 6)
* 4byte directive, Nios II: Nios II Directives. (line 22)
* 8byte directive, Nios II: Nios II Directives. (line 25)
* : (label): Statements. (line 31)
* @gotoff(SYMBOL), ARC modifier: ARC Modifiers. (line 20)
* @gotpc(SYMBOL), ARC modifier: ARC Modifiers. (line 16)
* @hi pseudo-op, XStormy16: XStormy16 Opcodes. (line 21)
* @lo pseudo-op, XStormy16: XStormy16 Opcodes. (line 10)
* @pcl(SYMBOL), ARC modifier: ARC Modifiers. (line 12)
* @plt(SYMBOL), ARC modifier: ARC Modifiers. (line 23)
* @sda(SYMBOL), ARC modifier: ARC Modifiers. (line 28)
* @word modifier, D10V: D10V-Word. (line 6)
* _ opcode prefix: Xtensa Opcodes. (line 9)
* __DYNAMIC__, ARC pre-defined symbol: ARC Symbols. (line 14)
* __GLOBAL_OFFSET_TABLE__, ARC pre-defined symbol: ARC Symbols.
                                         (line 11)
* a.out: Object. (line 6)
* a.out symbol attributes: a.out Symbols. (line 6)
* AArch64 floating point (IEEE): AArch64 Floating Point.
                                         (line 6)
* AArch64 immediate character: AArch64-Chars. (line 13)
* AArch64 line comment character: AArch64-Chars. (line 6)
* AArch64 line separator: AArch64-Chars. (line 10)
* AArch64 machine directives: AArch64 Directives. (line 6)
* AArch64 opcodes: AArch64 Opcodes. (line 6)
* AArch64 options (none): AArch64 Options. (line 6)
* AArch64 register names: AArch64-Regs. (line 6)
* AArch64 relocations: AArch64-Relocations.
                                         (line 6)
* AArch64 support: AArch64-Dependent. (line 6)
* ABI options, SH64: SH64 Options. (line 25)
* abort directive: Abort. (line 6)
* ABORT directive: ABORT (COFF). (line 6)
* absolute section: Ld Sections. (line 29)
* absolute-literals directive: Absolute Literals Directive.
                                         (line 6)
* ADDI instructions, relaxation: Xtensa Immediate Relaxation.
                                         (line 43)
* addition, permitted arguments: Infix Ops. (line 45)
* addresses: Expressions. (line 6)

```

* addresses, format of:	Secs Background.	(line 65)
* addressing modes, D10V:	D10V-Addressing.	(line 6)
* addressing modes, D30V:	D30V-Addressing.	(line 6)
* addressing modes, H8/300:	H8/300-Addressing.	(line 6)
* addressing modes, M680x0:	M68K-Syntax.	(line 21)
* addressing modes, M68HC11:	M68HC11-Syntax.	(line 29)
* addressing modes, SH:	SH-Addressing.	(line 6)
* addressing modes, SH64:	SH64-Addressing.	(line 6)
* addressing modes, XGATE:	XGATE-Syntax.	(line 28)
* addressing modes, Z8000:	Z8000-Addressing.	(line 6)
* ADR reg,<label> pseudo op, ARM:	ARM Opcodes.	(line 25)
* ADRL reg,<label> pseudo op, ARM:	ARM Opcodes.	(line 35)
* ADRP, ADD, LDR/STR group relocations, AArch64:	AArch64-Relocations.	(line 14)
* advancing location counter:	Org.	(line 6)
* align directive:	Align.	(line 6)
* align directive, Nios II:	Nios II Directives.	(line 6)
* align directive, SPARC:	Sparc-Directives.	(line 9)
* align directive, TIC54X:	TIC54X-Directives.	(line 6)
* aligned instruction bundle:	Bundle directives.	(line 9)
* alignment for NEON instructions:	ARM-Neon-Alignment.	(line 6)
* alignment of branch targets:	Xtensa Automatic Alignment.	(line 6)
* alignment of LOOP instructions:	Xtensa Automatic Alignment.	(line 6)
* Alpha floating point (IEEE):	Alpha Floating Point.	(line 6)
* Alpha line comment character:	Alpha-Chars.	(line 6)
* Alpha line separator:	Alpha-Chars.	(line 11)
* Alpha notes:	Alpha Notes.	(line 6)
* Alpha options:	Alpha Options.	(line 6)
* Alpha registers:	Alpha-Regs.	(line 6)
* Alpha relocations:	Alpha-Relocs.	(line 6)
* Alpha support:	Alpha-Dependent.	(line 6)
* Alpha Syntax:	Alpha Options.	(line 60)
* Alpha-only directives:	Alpha Directives.	(line 9)
* Altera Nios II support:	NiosII-Dependent.	(line 6)
* altered difference tables:	Word.	(line 12)
* alternate syntax for the 680x0:	M68K-Moto-Syntax.	(line 6)
* ARC Branch Target Address:	ARC-Regs.	(line 60)
* ARC BTA saved on exception entry:	ARC-Regs.	(line 79)
* ARC Build configuration for: BTA Registers:	ARC-Regs.	(line 89)
* ARC Build configuration for: Core Registers:	ARC-Regs.	(line 97)
* ARC Build configuration for: Interrupts:	ARC-Regs.	(line 93)
* ARC Build Configuration Registers Version:	ARC-Regs.	(line 85)
* ARC C preprocessor macro separator:	ARC-Chars.	(line 31)
* ARC core general registers:	ARC-Regs.	(line 10)
* ARC DCCM RAM Configuration Register:	ARC-Regs.	(line 101)
* ARC Exception Cause Register:	ARC-Regs.	(line 63)
* ARC Exception Return Address:	ARC-Regs.	(line 76)
* ARC extension core registers:	ARC-Regs.	(line 38)
* ARC frame pointer:	ARC-Regs.	(line 17)
* ARC global pointer:	ARC-Regs.	(line 14)
* ARC interrupt link register:	ARC-Regs.	(line 27)
* ARC Interrupt Vector Base address:	ARC-Regs.	(line 66)
* ARC level 1 interrupt link register:	ARC-Regs.	(line 23)
* ARC level 2 interrupt link register:	ARC-Regs.	(line 31)
* ARC line comment character:	ARC-Chars.	(line 11)
* ARC line separator:	ARC-Chars.	(line 27)

* ARC link register:	ARC-Regs.	(line 35)
* ARC loop counter:	ARC-Regs.	(line 41)
* ARC machine directives:	ARC Directives.	(line 6)
* ARC opcodes:	ARC Opcodes.	(line 6)
* ARC options:	ARC Options.	(line 6)
* ARC Processor Identification register:	ARC-Regs.	(line 51)
* ARC Program Counter:	ARC-Regs.	(line 54)
* ARC register name prefix character:	ARC-Chars.	(line 7)
* ARC register names:	ARC-Regs.	(line 6)
* ARC Saved User Stack Pointer:	ARC-Regs.	(line 73)
* ARC stack pointer:	ARC-Regs.	(line 20)
* ARC Status register:	ARC-Regs.	(line 57)
* ARC STATUS32 saved on exception:	ARC-Regs.	(line 82)
* ARC Stored STATUS32 register on entry to level P0 interrupts:	ARC-Regs.	(line 69)
* ARC support:	ARC-Dependent.	(line 6)
* ARC symbol prefix character:	ARC-Chars.	(line 20)
* ARC word aligned program counter:	ARC-Regs.	(line 44)
* arch directive, i386:	i386-Arch.	(line 6)
* arch directive, M680x0:	M68K-Directives.	(line 22)
* arch directive, MSP 430:	MSP430 Directives.	(line 18)
* arch directive, x86-64:	i386-Arch.	(line 6)
* architecture options, i960:	Options-i960.	(line 6)
* architecture options, IP2022:	IP2K-Opts.	(line 9)
* architecture options, IP2K:	IP2K-Opts.	(line 14)
* architecture options, M16C:	M32C-Opts.	(line 12)
* architecture options, M32C:	M32C-Opts.	(line 9)
* architecture options, M32R:	M32R-Opts.	(line 21)
* architecture options, M32R2:	M32R-Opts.	(line 17)
* architecture options, M32RX:	M32R-Opts.	(line 9)
* architecture options, M680x0:	M68K-Opts.	(line 99)
* Architecture variant option, CRIS:	CRIS-Opts.	(line 34)
* architectures, Meta:	Meta Options.	(line 6)
* architectures, PowerPC:	PowerPC-Opts.	(line 6)
* architectures, SCORE:	SCORE-Opts.	(line 6)
* architectures, SPARC:	Sparc-Opts.	(line 6)
* arguments for addition:	Infix Ops.	(line 45)
* arguments for subtraction:	Infix Ops.	(line 50)
* arguments in expressions:	Arguments.	(line 6)
* arithmetic functions:	Operators.	(line 6)
* arithmetic operands:	Arguments.	(line 6)
* ARM data relocations:	ARM-Relocations.	(line 6)
* ARM floating point (IEEE):	ARM Floating Point.	(line 6)
* ARM identifiers:	ARM-Chars.	(line 19)
* ARM immediate character:	ARM-Chars.	(line 17)
* ARM line comment character:	ARM-Chars.	(line 6)
* ARM line separator:	ARM-Chars.	(line 14)
* ARM machine directives:	ARM Directives.	(line 6)
* ARM opcodes:	ARM Opcodes.	(line 6)
* ARM options (none):	ARM Options.	(line 6)
* ARM register names:	ARM-Regs.	(line 6)
* ARM support:	ARM-Dependent.	(line 6)
* ascii directive:	Ascii.	(line 6)
* asciz directive:	Asciz.	(line 6)
* asg directive, TIC54X:	TIC54X-Directives.	(line 18)
* assembler bugs, reporting:	Bug Reporting.	(line 6)
* assembler crash:	Bug Criteria.	(line 9)
* assembler directive .3byte, RX:	RX-Directives.	(line 9)
* assembler directive .arch, CRIS:	CRIS-Pseudos.	(line 50)

```
* assembler directive .dword, CRIS: CRIS-Pseudos. (line 12)
* assembler directive .far, M68HC11: M68HC11-Directives. (line 20)
* assembler directive .fetchalign, RX: RX-Directives. (line 13)
* assembler directive .interrupt, M68HC11: M68HC11-Directives.
                                         (line 26)
* assembler directive .mode, M68HC11: M68HC11-Directives. (line 16)
* assembler directive .relax, M68HC11: M68HC11-Directives. (line 10)
* assembler directive .syntax, CRIS: CRIS-Pseudos. (line 18)
* assembler directive .xrefb, M68HC11: M68HC11-Directives. (line 31)
* assembler directive BSPEC, MMIX: MMIX-Pseudos. (line 137)
* assembler directive BYTE, MMIX: MMIX-Pseudos. (line 101)
* assembler directive ESPEC, MMIX: MMIX-Pseudos. (line 137)
* assembler directive GREG, MMIX: MMIX-Pseudos. (line 53)
* assembler directive IS, MMIX: MMIX-Pseudos. (line 44)
* assembler directive LOC, MMIX: MMIX-Pseudos. (line 7)
* assembler directive LOCAL, MMIX: MMIX-Pseudos. (line 29)
* assembler directive OCTA, MMIX: MMIX-Pseudos. (line 113)
* assembler directive PREFIX, MMIX: MMIX-Pseudos. (line 125)
* assembler directive TETRA, MMIX: MMIX-Pseudos. (line 113)
* assembler directive WYDE, MMIX: MMIX-Pseudos. (line 113)
* assembler directives, CRIS: CRIS-Pseudos. (line 6)
* assembler directives, M68HC11: M68HC11-Directives. (line 6)
* assembler directives, M68HC12: M68HC11-Directives. (line 6)
* assembler directives, MMIX: MMIX-Pseudos. (line 6)
* assembler directives, RL78: RL78-Directives. (line 6)
* assembler directives, RX: RX-Directives. (line 6)
* assembler directives, XGATE: XGATE-Directives. (line 6)
* assembler internal logic error: As Sections. (line 13)
* assembler version: v. (line 6)
* assembler, and linker: Secs Background. (line 10)
* assembly listings, enabling: a. (line 6)
* assigning values to symbols: Setting Symbols. (line 6)
* assigning values to symbols <1>: Equ. (line 6)
* at register, MIPS: MIPS Macros. (line 35)
* atmp directive, i860: Directives-i860. (line 16)
* attributes, symbol: Symbol Attributes. (line 6)
* att_syntax pseudo op, i386: i386-Variations. (line 6)
* att_syntax pseudo op, x86-64: i386-Variations. (line 6)
* auxiliary attributes, COFF symbols: COFF Symbols. (line 19)
* auxiliary symbol information, COFF: Dim. (line 6)
* AVR line comment character: AVR-Chars. (line 6)
* AVR line separator: AVR-Chars. (line 14)
* AVR modifiers: AVR-Modifiers. (line 6)
* AVR opcode summary: AVR Opcodes. (line 6)
* AVR options (none): AVR Options. (line 6)
* AVR register names: AVR-Regs. (line 6)
* AVR support: AVR-Dependent. (line 6)
* A_DIR environment variable, TIC54X: TIC54X-Env. (line 6)
* backslash (\): Strings. (line 40)
* backspace (\b): Strings. (line 15)
* balign directive: Balign. (line 6)
* balignl directive: Balign. (line 27)
* balignw directive: Balign. (line 27)
* bes directive, TIC54X: TIC54X-Directives. (line 194)
* big endian output, MIPS: Overview. (line 806)
* big endian output, PJ: Overview. (line 713)
* big-endian output, MIPS: MIPS Options. (line 13)
* big-endian output, TIC6X: TIC6X Options. (line 46)
* bignums: Bignums. (line 6)
```

* binary constants, TIC54X:	TIC54X-Constants.	(line 8)
* binary files, including:	Incbin.	(line 6)
* binary integers:	Integers.	(line 6)
* bit names, IA-64:	IA-64-Bits.	(line 6)
* bitfields, not supported on VAX:	VAX-no.	(line 6)
* Blackfin directives:	Blackfin Directives.	(line 6)
* Blackfin options (none):	Blackfin Options.	(line 6)
* Blackfin support:	Blackfin-Dependent.	(line 6)
* Blackfin syntax:	Blackfin Syntax.	(line 6)
* block:	Z8000 Directives.	(line 55)
* BMI, i386:	i386-BMI.	(line 6)
* BMI, x86-64:	i386-BMI.	(line 6)
* branch improvement, M680x0:	M68K-Branch.	(line 6)
* branch improvement, M68HC11:	M68HC11-Branch.	(line 6)
* branch improvement, VAX:	VAX-branch.	(line 6)
* branch instructions, relaxation:	Xtensa Branch Relaxation.	(line 6)
* branch recording, i960:	Options-i960.	(line 22)
* branch statistics table, i960:	Options-i960.	(line 40)
* Branch Target Address, ARC:	ARC-Regs.	(line 60)
* branch target alignment:	Xtensa Automatic Alignment.	(line 6)
* break directive, TIC54X:	TIC54X-Directives.	(line 141)
* BSD syntax:	PDP-11-Syntax.	(line 6)
* bss directive, i960:	Directives-i960.	(line 6)
* bss directive, TIC54X:	TIC54X-Directives.	(line 27)
* bss section:	Ld Sections.	(line 20)
* bss section <1>:	bss.	(line 6)
* BTA saved on exception entry, ARC:	ARC-Regs.	(line 79)
* bug criteria:	Bug Criteria.	(line 6)
* bug reports:	Bug Reporting.	(line 6)
* bugs in assembler:	Reporting Bugs.	(line 6)
* Build configuration for: BTA Registers, ARC:	ARC-Regs.	(line 89)
* Build configuration for: Core Registers, ARC:	ARC-Regs.	(line 97)
* Build configuration for: Interrupts, ARC:	ARC-Regs.	(line 93)
* Build Configuration Registers Version, ARC:	ARC-Regs.	(line 85)
* Built-in symbols, CRIS:	CRIS-Symbols.	(line 6)
* builtin math functions, TIC54X:	TIC54X-Builtins.	(line 6)
* builtin subsym functions, TIC54X:	TIC54X-Macros.	(line 16)
* bundle:	Bundle directives.	(line 9)
* bundle-locked:	Bundle directives.	(line 39)
* bundle_align_mode directive:	Bundle directives.	(line 9)
* bundle_lock directive:	Bundle directives.	(line 31)
* bundle_unlock directive:	Bundle directives.	(line 31)
* bus lock prefixes, i386:	i386-Prefixes.	(line 36)
* bval:	Z8000 Directives.	(line 30)
* byte directive:	Byte.	(line 6)
* byte directive, TIC54X:	TIC54X-Directives.	(line 34)
* C preprocessor macro separator, ARC:	ARC-Chars.	(line 31)
* C54XDSP_DIR environment variable, TIC54X:	TIC54X-Env.	(line 6)
* call directive, Nios II:	Nios II Relocations.	(line 38)
* call instructions, i386:	i386-Mnemonics.	(line 58)
* call instructions, relaxation:	Xtensa Call Relaxation.	(line 6)
* call instructions, x86-64:	i386-Mnemonics.	(line 58)
* callj, i960 pseudo-opcode:	callj-i960.	(line 6)
* call_hiadj directive, Nios II:	Nios II Relocations.	

		(line 38)
* call_lo directive, Nios II:	Nios II Relocations.	(line 38)
* carriage return (backslash-r):	Strings.	(line 24)
* case sensitivity, Z80:	Z80-Case.	(line 6)
* cfi_endproc directive:	CFI directives.	(line 40)
* cfi_fde_data directive:	CFI directives.	(line 66)
* cfi_personality directive:	CFI directives.	(line 47)
* cfi_personality_id directive:	CFI directives.	(line 59)
* cfi_sections directive:	CFI directives.	(line 9)
* cfi_startproc directive:	CFI directives.	(line 30)
* char directive, TIC54X:	TIC54X-Directives.	(line 34)
* character constant, Z80:	Z80-Chars.	(line 20)
* character constants:	Characters.	(line 6)
* character escape codes:	Strings.	(line 15)
* character escapes, Z80:	Z80-Chars.	(line 18)
* character, single:	Chars.	(line 6)
* characters used in symbols:	Symbol Intro.	(line 6)
* clink directive, TIC54X:	TIC54X-Directives.	(line 43)
* code16 directive, i386:	i386-16bit.	(line 6)
* code16gcc directive, i386:	i386-16bit.	(line 6)
* code32 directive, i386:	i386-16bit.	(line 6)
* code64 directive, i386:	i386-16bit.	(line 6)
* code64 directive, x86-64:	i386-16bit.	(line 6)
* COFF auxiliary symbol information:	Dim.	(line 6)
* COFF structure debugging:	Tag.	(line 6)
* COFF symbol attributes:	COFF Symbols.	(line 6)
* COFF symbol descriptor:	Desc.	(line 6)
* COFF symbol storage class:	Scl.	(line 6)
* COFF symbol type:	Type.	(line 11)
* COFF symbols, debugging:	Def.	(line 6)
* COFF value attribute:	Val.	(line 6)
* COMDAT:	Linkonce.	(line 6)
* comm directive:	Comm.	(line 6)
* command line conventions:	Command Line.	(line 6)
* command line options, V850:	V850 Options.	(line 9)
* command-line options ignored, VAX:	VAX-Opts.	(line 6)
* comment character, XStormy16:	XStormy16-Chars.	(line 11)
* comments:	Comments.	(line 6)
* comments, M680x0:	M68K-Chars.	(line 6)
* comments, removed by preprocessor:	Preprocessing.	(line 11)
* common directive, SPARC:	Sparc-Directives.	(line 12)
* common sections:	Linkonce.	(line 6)
* common variable storage:	bss.	(line 6)
* compare and jump expansions, i960:	Compare-and-branch-i960.	(line 13)
* compare/branch instructions, i960:	Compare-and-branch-i960.	(line 6)
* comparison expressions:	Infix Ops.	(line 56)
* conditional assembly:	If.	(line 6)
* constant, single character:	Chars.	(line 6)
* constants:	Constants.	(line 6)
* constants, bignum:	Bignums.	(line 6)
* constants, character:	Characters.	(line 6)
* constants, converted by preprocessor:	Preprocessing.	(line 14)
* constants, floating point:	Flonums.	(line 6)
* constants, integer:	Integers.	(line 6)
* constants, number:	Numbers.	(line 6)
* constants, Sparc:	Sparc-Constants.	(line 6)

* constants, string:	Strings.	(line 6)
* constants, TIC54X:	TIC54X-Constants.	(line 6)
* conversion instructions, i386:	i386-Mnemonics.	(line 39)
* conversion instructions, x86-64:	i386-Mnemonics.	(line 39)
* coprocessor wait, i386:	i386-Prefixes.	(line 40)
* copy directive, TIC54X:	TIC54X-Directives.	(line 52)
* core general registers, ARC:	ARC-Regs.	(line 10)
* cpu directive, ARC:	ARC Directives.	(line 27)
* cpu directive, M680x0:	M68K-Directives.	(line 30)
* cpu directive, MSP 430:	MSP430 Directives.	(line 22)
* CR16 line comment character:	CR16-Chars.	(line 6)
* CR16 line separator:	CR16-Chars.	(line 12)
* CR16 Operand Qualifiers:	CR16 Operand Qualifiers.	(line 6)
* CR16 support:	CR16-Dependent.	(line 6)
* crash of assembler:	Bug Criteria.	(line 9)
* CRIS --emulation=crisaout command line option:	CRIS-Opts.	(line 9)
* CRIS --emulation=criself command line option:	CRIS-Opts.	(line 9)
* CRIS --march=ARCHITECTURE command line option:	CRIS-Opts.	(line 34)
* CRIS --mul-bug-abort command line option:	CRIS-Opts.	(line 63)
* CRIS --no-mul-bug-abort command line option:	CRIS-Opts.	(line 63)
* CRIS --no-underscore command line option:	CRIS-Opts.	(line 15)
* CRIS --pic command line option:	CRIS-Opts.	(line 27)
* CRIS --underscore command line option:	CRIS-Opts.	(line 15)
* CRIS -N command line option:	CRIS-Opts.	(line 59)
* CRIS architecture variant option:	CRIS-Opts.	(line 34)
* CRIS assembler directive .arch:	CRIS-Pseudos.	(line 50)
* CRIS assembler directive .dword:	CRIS-Pseudos.	(line 12)
* CRIS assembler directive .syntax:	CRIS-Pseudos.	(line 18)
* CRIS assembler directives:	CRIS-Pseudos.	(line 6)
* CRIS built-in symbols:	CRIS-Symbols.	(line 6)
* CRIS instruction expansion:	CRIS-Expand.	(line 6)
* CRIS line comment characters:	CRIS-Chars.	(line 6)
* CRIS options:	CRIS-Opts.	(line 6)
* CRIS position-independent code:	CRIS-Opts.	(line 27)
* CRIS pseudo-op .arch:	CRIS-Pseudos.	(line 50)
* CRIS pseudo-op .dword:	CRIS-Pseudos.	(line 12)
* CRIS pseudo-op .syntax:	CRIS-Pseudos.	(line 18)
* CRIS pseudo-ops:	CRIS-Pseudos.	(line 6)
* CRIS register names:	CRIS-Regs.	(line 6)
* CRIS support:	CRIS-Dependent.	(line 6)
* CRIS symbols in position-independent code:	CRIS-Pic.	(line 6)
* ctbp register, V850:	V850-Regs.	(line 90)
* ctoff pseudo-op, V850:	V850 Opcodes.	(line 110)
* ctpc register, V850:	V850-Regs.	(line 82)
* ctpsw register, V850:	V850-Regs.	(line 84)
* current address:	Dot.	(line 6)
* current address, advancing:	Org.	(line 6)
* c_mode directive, TIC54X:	TIC54X-Directives.	(line 49)
* D10V @word modifier:	D10V-Word.	(line 6)
* D10V addressing modes:	D10V-Addressing.	(line 6)
* D10V floating point:	D10V-Float.	(line 6)
* D10V line comment character:	D10V-Chars.	(line 6)
* D10V opcode summary:	D10V-Opcodes.	(line 6)
* D10V optimization:	Overview.	(line 579)
* D10V options:	D10V-Opts.	(line 6)
* D10V registers:	D10V-Regs.	(line 6)
* D10V size modifiers:	D10V-Size.	(line 6)
* D10V sub-instruction ordering:	D10V-Chars.	(line 14)

* D10V sub-instructions:	D10V-Subs.	(line 6)
* D10V support:	D10V-Dependent.	(line 6)
* D10V syntax:	D10V-Syntax.	(line 6)
* D30V addressing modes:	D30V-Addressing.	(line 6)
* D30V floating point:	D30V-Float.	(line 6)
* D30V Guarded Execution:	D30V-Guarded.	(line 6)
* D30V line comment character:	D30V-Chars.	(line 6)
* D30V nops:	Overview.	(line 587)
* D30V nops after 32-bit multiply:	Overview.	(line 590)
* D30V opcode summary:	D30V-Opcodes.	(line 6)
* D30V optimization:	Overview.	(line 584)
* D30V options:	D30V-Opts.	(line 6)
* D30V registers:	D30V-Regs.	(line 6)
* D30V size modifiers:	D30V-Size.	(line 6)
* D30V sub-instruction ordering:	D30V-Chars.	(line 14)
* D30V sub-instructions:	D30V-Subs.	(line 6)
* D30V support:	D30V-Dependent.	(line 6)
* D30V syntax:	D30V-Syntax.	(line 6)
* data alignment on SPARC:	Sparc-Aligned-Data.	(line 6)
* data and text sections, joining:	R.	(line 6)
* data directive:	Data.	(line 6)
* data directive, TIC54X:	TIC54X-Directives.	(line 59)
* data relocations, ARM:	ARM-Relocations.	(line 6)
* data section:	Ld Sections.	(line 9)
* data1 directive, M680x0:	M68K-Directives.	(line 9)
* data2 directive, M680x0:	M68K-Directives.	(line 12)
* datalabel, SH64:	SH64-Addressing.	(line 16)
* dbpc register, V850:	V850-Regs.	(line 86)
* dbpsw register, V850:	V850-Regs.	(line 88)
* DCCM RAM Configuration Register, ARC:	ARC-Regs.	(line 101)
* debuggers, and symbol order:	Symbols.	(line 10)
* debugging COFF symbols:	Def.	(line 6)
* DEC syntax:	PDP-11-Syntax.	(line 6)
* decimal integers:	Integers.	(line 12)
* def directive:	Def.	(line 6)
* def directive, TIC54X:	TIC54X-Directives.	(line 101)
* density instructions:	Density Instructions.	(line 6)
* dependency tracking:	MD.	(line 6)
* deprecated directives:	Deprecated.	(line 6)
* desc directive:	Desc.	(line 6)
* descriptor, of a.out symbol:	Symbol Desc.	(line 6)
* dfloat directive, VAX:	VAX-directives.	(line 9)
* difference tables altered:	Word.	(line 12)
* difference tables, warning:	K.	(line 6)
* differences, mmixal:	MMIX-mmixal.	(line 6)
* dim directive:	Dim.	(line 6)
* directives and instructions:	Statements.	(line 20)
* directives for PowerPC:	PowerPC-Pseudo.	(line 6)
* directives for SCORE:	SCORE-Pseudo.	(line 6)
* directives, Blackfin:	Blackfin Directives.	(line 6)
* directives, M32R:	M32R-Directives.	(line 6)
* directives, M680x0:	M68K-Directives.	(line 6)
* directives, machine independent:	Pseudo Ops.	(line 6)
* directives, Xtensa:	Xtensa Directives.	(line 6)
* directives, Z8000:	Z8000 Directives.	(line 6)
* Disable floating-point instructions:	MIPS Floating-Point.	(line 6)

```

* Disable single-precision floating-point operations: MIPS Floating-Point.
                                                                    (line 12)
* displacement sizing character, VAX:      VAX-operands.      (line 12)
* dollar local symbols:                   Symbol Names.      (line 113)
* dot (symbol):                           Dot.              (line 6)
* double directive:                       Double.           (line 6)
* double directive, i386:                  i386-Float.      (line 14)
* double directive, M680x0:                M68K-Float.      (line 14)
* double directive, M68HC11:              M68HC11-Float.   (line 14)
* double directive, RX:                   RX-Float.        (line 11)
* double directive, TIC54X:               TIC54X-Directives. (line 62)
* double directive, VAX:                  VAX-float.       (line 15)
* double directive, x86-64:               i386-Float.      (line 14)
* double directive, XGATE:                XGATE-Float.     (line 13)
* doublequote ("\"):                      Strings.          (line 43)
* drlist directive, TIC54X:              TIC54X-Directives. (line 71)
* drnolist directive, TIC54X:            TIC54X-Directives. (line 71)
* dual directive, i860:                   Directives-i860.  (line 6)
* dword directive, Nios II:               Nios II Directives. (line 16)
* EB command line option, Nios II:        Nios II Options.  (line 22)
* ecr register, V850:                     V850-Regs.       (line 78)
* eight-byte integer:                     Quad.            (line 9)
* eipc register, V850:                     V850-Regs.       (line 70)
* eipsw register, V850:                   V850-Regs.       (line 72)
* eject directive:                        Eject.           (line 6)
* EL command line option, Nios II:        Nios II Options.  (line 25)
* ELF symbol type:                         Type.            (line 22)
* else directive:                          Else.            (line 6)
* elseif directive:                       Elseif.          (line 6)
* empty expressions:                      Empty Exprs.     (line 6)
* emsg directive, TIC54X:                 TIC54X-Directives. (line 75)
* emulation:                              Overview.        (line 1007)
* encoding options, i386:                 i386-Mnemonics.  (line 34)
* encoding options, x86-64:              i386-Mnemonics.  (line 34)
* end directive:                          End.             (line 6)
* enddual directive, i860:                 Directives-i860.  (line 11)
* endif directive:                        Endif.           (line 6)
* endfunc directive:                      Endfunc.         (line 6)
* endianness, MIPS:                       Overview.        (line 806)
* endianness, PJ:                         Overview.        (line 713)
* endif directive:                        Endif.           (line 6)
* endloop directive, TIC54X:              TIC54X-Directives. (line 141)
* endm directive:                          Macro.           (line 137)
* endm directive, TIC54X:                 TIC54X-Directives. (line 151)
* endstruct directive, TIC54X:            TIC54X-Directives. (line 214)
* endunion directive, TIC54X:             TIC54X-Directives. (line 248)
* environment settings, TIC54X:          TIC54X-Env.      (line 6)
* EOF, newline must precede:              Statements.      (line 14)
* ep register, V850:                       V850-Regs.       (line 66)
* Epiphany line comment character:        Epiphany-Chars.  (line 6)
* Epiphany line separator:                Epiphany-Chars.  (line 14)
* Epiphany options:                       Epiphany Options. (line 6)
* Epiphany support:                       Epiphany-Dependent. (line 6)
* equ directive:                           Equ.             (line 6)
* equ directive, TIC54X:                  TIC54X-Directives. (line 189)
* equiv directive:                         Equiv.           (line 6)
* eqv directive:                           Eqv.            (line 6)
* err directive:                           Err.            (line 6)
* error directive:                         Error.           (line 6)

```

* error messages:	Errors.	(line 6)
* error on valid input:	Bug Criteria.	(line 12)
* errors, caused by warnings:	W.	(line 16)
* errors, continuing after:	Z.	(line 6)
* ESA/390 floating point (IEEE):	ESA/390 Floating Point.	(line 6)
* ESA/390 support:	ESA/390-Dependent.	(line 6)
* ESA/390 Syntax:	ESA/390 Options.	(line 7)
* ESA/390-only directives:	ESA/390 Directives.	(line 12)
* escape codes, character:	Strings.	(line 15)
* eval directive, TIC54X:	TIC54X-Directives.	(line 22)
* even:	Z8000 Directives.	(line 58)
* even directive, M680x0:	M68K-Directives.	(line 15)
* even directive, TIC54X:	TIC54X-Directives.	(line 6)
* Exception Cause Register, ARC:	ARC-Regs.	(line 63)
* Exception Return Address, ARC:	ARC-Regs.	(line 76)
* exitm directive:	Macro.	(line 140)
* expr (internal section):	As Sections.	(line 17)
* expression arguments:	Arguments.	(line 6)
* expressions:	Expressions.	(line 6)
* expressions, comparison:	Infix Ops.	(line 56)
* expressions, empty:	Empty Exprs.	(line 6)
* expressions, integer:	Integer Exprs.	(line 6)
* extAuxRegister directive, ARC:	ARC Directives.	(line 105)
* extCondCode directive, ARC:	ARC Directives.	(line 126)
* extCoreRegister directive, ARC:	ARC Directives.	(line 137)
* extend directive M680x0:	M68K-Float.	(line 17)
* extend directive M68HC11:	M68HC11-Float.	(line 17)
* extend directive XGATE:	XGATE-Float.	(line 16)
* extended directive, i960:	Directives-i960.	(line 13)
* extension core registers, ARC:	ARC-Regs.	(line 38)
* extern directive:	Extern.	(line 6)
* extInstruction directive, ARC:	ARC Directives.	(line 164)
* fail directive:	Fail.	(line 6)
* far_mode directive, TIC54X:	TIC54X-Directives.	(line 80)
* faster processing (-f):	f.	(line 6)
* fatal signal:	Bug Criteria.	(line 9)
* fclist directive, TIC54X:	TIC54X-Directives.	(line 85)
* fcnolist directive, TIC54X:	TIC54X-Directives.	(line 85)
* fepc register, V850:	V850-Regs.	(line 74)
* fepsw register, V850:	V850-Regs.	(line 76)
* ffloat directive, VAX:	VAX-directives.	(line 13)
* field directive, TIC54X:	TIC54X-Directives.	(line 89)
* file directive:	File.	(line 6)
* file directive, MSP 430:	MSP430 Directives.	(line 6)
* file name, logical:	File.	(line 13)
* file names and line numbers, in warnings/errors:	Errors.	(line 16)
* files, including:	Include.	(line 6)
* files, input:	Input Files.	(line 6)
* fill directive:	Fill.	(line 6)
* filling memory:	Skip.	(line 6)
* filling memory <1>:	Space.	(line 6)
* filling memory with zero bytes:	Zero.	(line 6)
* FLIX syntax:	Xtensa Syntax.	(line 6)
* float directive:	Float.	(line 6)
* float directive, i386:	i386-Float.	(line 14)
* float directive, M680x0:	M68K-Float.	(line 11)
* float directive, M68HC11:	M68HC11-Float.	(line 11)
* float directive, RX:	RX-Float.	(line 8)

* float directive, TIC54X:	TIC54X-Directives.	(line 62)
* float directive, VAX:	VAX-float.	(line 15)
* float directive, x86-64:	i386-Float.	(line 14)
* float directive, XGATE:	XGATE-Float.	(line 10)
* floating point numbers:	Flonums.	(line 6)
* floating point numbers (double):	Double.	(line 6)
* floating point numbers (single):	Float.	(line 6)
* floating point numbers (single) <1>:	Single.	(line 6)
* floating point, AArch64 (IEEE):	AArch64 Floating Point.	(line 6)
* floating point, Alpha (IEEE):	Alpha Floating Point.	(line 6)
* floating point, ARM (IEEE):	ARM Floating Point.	(line 6)
* floating point, D10V:	D10V-Float.	(line 6)
* floating point, D30V:	D30V-Float.	(line 6)
* floating point, ESA/390 (IEEE):	ESA/390 Floating Point.	(line 6)
* floating point, H8/300 (IEEE):	H8/300 Floating Point.	(line 6)
* floating point, HPPA (IEEE):	HPPA Floating Point.	(line 6)
* floating point, i386:	i386-Float.	(line 6)
* floating point, i960 (IEEE):	Floating Point-i960.	(line 6)
* floating point, M680x0:	M68K-Float.	(line 6)
* floating point, M68HC11:	M68HC11-Float.	(line 6)
* floating point, MSP 430 (IEEE):	MSP430 Floating Point.	(line 6)
* floating point, RX:	RX-Float.	(line 6)
* floating point, s390:	s390 Floating Point.	(line 6)
* floating point, SH (IEEE):	SH Floating Point.	(line 6)
* floating point, SPARC (IEEE):	Sparc-Float.	(line 6)
* floating point, V850 (IEEE):	V850 Floating Point.	(line 6)
* floating point, VAX:	VAX-float.	(line 6)
* floating point, x86-64:	i386-Float.	(line 6)
* floating point, XGATE:	XGATE-Float.	(line 6)
* floating point, Z80:	Z80 Floating Point.	(line 6)
* flonums:	Flonums.	(line 6)
* format of error messages:	Errors.	(line 38)
* format of warning messages:	Errors.	(line 12)
* formfeed (\f):	Strings.	(line 18)
* frame pointer, ARC:	ARC-Regs.	(line 17)
* func directive:	Func.	(line 6)
* functions, in expressions:	Operators.	(line 6)
* gbr960, i960 postprocessor:	Options-i960.	(line 40)
* gfloat directive, VAX:	VAX-directives.	(line 17)
* global:	Z8000 Directives.	(line 21)
* global directive:	Global.	(line 6)
* global directive, TIC54X:	TIC54X-Directives.	(line 101)
* global pointer, ARC:	ARC-Regs.	(line 14)
* got directive, Nios II:	Nios II Relocations.	(line 38)
* gotoff directive, Nios II:	Nios II Relocations.	(line 38)
* gotoff_hiadj directive, Nios II:	Nios II Relocations.	(line 38)
* gotoff_lo directive, Nios II:	Nios II Relocations.	(line 38)

		(line 38)
* got_hiadj directive, Nios II:	Nios II Relocations.	(line 38)
* got_lo directive, Nios II:	Nios II Relocations.	(line 38)
* gp register, MIPS:	MIPS Small Data.	(line 6)
* gp register, V850:	V850-Regs.	(line 14)
* gprel directive, Nios II:	Nios II Relocations.	(line 26)
* grouping data:	Sub-Sections.	(line 6)
* H8/300 addressing modes:	H8/300-Addressing.	(line 6)
* H8/300 floating point (IEEE):	H8/300 Floating Point.	(line 6)
* H8/300 line comment character:	H8/300-Chars.	(line 6)
* H8/300 line separator:	H8/300-Chars.	(line 8)
* H8/300 machine directives (none):	H8/300 Directives.	(line 6)
* H8/300 opcode summary:	H8/300 Opcodes.	(line 6)
* H8/300 options:	H8/300 Options.	(line 6)
* H8/300 registers:	H8/300-Regs.	(line 6)
* H8/300 size suffixes:	H8/300 Opcodes.	(line 160)
* H8/300 support:	H8/300-Dependent.	(line 6)
* H8/300H, assembling for:	H8/300 Directives.	(line 8)
* half directive, Nios II:	Nios II Directives.	(line 10)
* half directive, SPARC:	Sparc-Directives.	(line 17)
* half directive, TIC54X:	TIC54X-Directives.	(line 109)
* hex character code (\XD.):	Strings.	(line 36)
* hexadecimal integers:	Integers.	(line 15)
* hexadecimal prefix, Z80:	Z80-Chars.	(line 15)
* hfloat directive, VAX:	VAX-directives.	(line 21)
* hi directive, Nios II:	Nios II Relocations.	(line 20)
* hi pseudo-op, V850:	V850 Opcodes.	(line 33)
* hi0 pseudo-op, V850:	V850 Opcodes.	(line 10)
* hiadj directive, Nios II:	Nios II Relocations.	(line 6)
* hidden directive:	Hidden.	(line 6)
* high directive, M32R:	M32R-Directives.	(line 18)
* hilo pseudo-op, V850:	V850 Opcodes.	(line 55)
* HPPA directives not supported:	HPPA Directives.	(line 11)
* HPPA floating point (IEEE):	HPPA Floating Point.	(line 6)
* HPPA Syntax:	HPPA Options.	(line 7)
* HPPA-only directives:	HPPA Directives.	(line 24)
* hword directive:	hword.	(line 6)
* i370 support:	ESA/390-Dependent.	(line 6)
* i386 16-bit code:	i386-16bit.	(line 6)
* i386 arch directive:	i386-Arch.	(line 6)
* i386 att_syntax pseudo op:	i386-Variations.	(line 6)
* i386 conversion instructions:	i386-Mnemonics.	(line 39)
* i386 floating point:	i386-Float.	(line 6)
* i386 immediate operands:	i386-Variations.	(line 15)
* i386 instruction naming:	i386-Mnemonics.	(line 9)
* i386 instruction prefixes:	i386-Prefixes.	(line 6)
* i386 intel_syntax pseudo op:	i386-Variations.	(line 6)
* i386 jump optimization:	i386-Jumps.	(line 6)
* i386 jump, call, return:	i386-Variations.	(line 40)
* i386 jump/call operands:	i386-Variations.	(line 15)
* i386 line comment character:	i386-Chars.	(line 6)
* i386 line separator:	i386-Chars.	(line 18)

* i386 memory references:	i386-Memory.	(line 6)
* i386 mnemonic compatibility:	i386-Mnemonics.	(line 64)
* i386 mul, imul instructions:	i386-Notes.	(line 6)
* i386 options:	i386-Options.	(line 6)
* i386 register operands:	i386-Variations.	(line 15)
* i386 registers:	i386-Regs.	(line 6)
* i386 sections:	i386-Variations.	(line 46)
* i386 size suffixes:	i386-Variations.	(line 28)
* i386 source, destination operands:	i386-Variations.	(line 21)
* i386 support:	i386-Dependent.	(line 6)
* i386 syntax compatibility:	i386-Variations.	(line 6)
* i80386 support:	i386-Dependent.	(line 6)
* i860 line comment character:	i860-Chars.	(line 6)
* i860 line separator:	i860-Chars.	(line 14)
* i860 machine directives:	Directives-i860.	(line 6)
* i860 opcodes:	Opcodes for i860.	(line 6)
* i860 support:	i860-Dependent.	(line 6)
* i960 architecture options:	Options-i960.	(line 6)
* i960 branch recording:	Options-i960.	(line 22)
* i960 callj pseudo-opcode:	callj-i960.	(line 6)
* i960 compare and jump expansions:	Compare-and-branch-i960.	(line 13)
* i960 compare/branch instructions:	Compare-and-branch-i960.	(line 6)
* i960 floating point (IEEE):	Floating Point-i960.	(line 6)
* i960 line comment character:	i960-Chars.	(line 6)
* i960 line separator:	i960-Chars.	(line 14)
* i960 machine directives:	Directives-i960.	(line 6)
* i960 opcodes:	Opcodes for i960.	(line 6)
* i960 options:	Options-i960.	(line 6)
* i960 support:	i960-Dependent.	(line 6)
* IA-64 line comment character:	IA-64-Chars.	(line 6)
* IA-64 line separator:	IA-64-Chars.	(line 8)
* IA-64 options:	IA-64 Options.	(line 6)
* IA-64 Processor-status-Register bit names:	IA-64-Bits.	(line 6)
* IA-64 registers:	IA-64-Regs.	(line 6)
* IA-64 relocations:	IA-64-Relocs.	(line 6)
* IA-64 support:	IA-64-Dependent.	(line 6)
* IA-64 Syntax:	IA-64 Options.	(line 85)
* ident directive:	Ident.	(line 6)
* identifiers, ARM:	ARM-Chars.	(line 19)
* identifiers, MSP 430:	MSP430-Chars.	(line 17)
* if directive:	If.	(line 6)
* ifb directive:	If.	(line 21)
* ifc directive:	If.	(line 25)
* ifdef directive:	If.	(line 16)
* ifeq directive:	If.	(line 33)
* ifeqs directive:	If.	(line 36)
* ifge directive:	If.	(line 40)
* ifgt directive:	If.	(line 44)
* ifle directive:	If.	(line 48)
* iflt directive:	If.	(line 52)
* ifnb directive:	If.	(line 56)
* ifnc directive:	If.	(line 61)
* ifndef directive:	If.	(line 65)
* ifne directive:	If.	(line 72)
* ifnes directive:	If.	(line 76)
* ifnotdef directive:	If.	(line 65)

* immediate character, AArch64:	AArch64-Chars.	(line 13)
* immediate character, ARM:	ARM-Chars.	(line 17)
* immediate character, M680x0:	M68K-Chars.	(line 13)
* immediate character, VAX:	VAX-operands.	(line 6)
* immediate fields, relaxation:	Xtensa Immediate Relaxation.	(line 6)
* immediate operands, i386:	i386-Variations.	(line 15)
* immediate operands, x86-64:	i386-Variations.	(line 15)
* imul instruction, i386:	i386-Notes.	(line 6)
* imul instruction, x86-64:	i386-Notes.	(line 6)
* incbin directive:	Incbin.	(line 6)
* include directive:	Include.	(line 6)
* include directive search path:	I.	(line 6)
* indirect character, VAX:	VAX-operands.	(line 9)
* infix operators:	Infix Ops.	(line 6)
* inhibiting interrupts, i386:	i386-Prefixes.	(line 36)
* input:	Input Files.	(line 6)
* input file linenumbers:	Input Files.	(line 35)
* instruction aliases, s390:	s390 Aliases.	(line 6)
* instruction bundle:	Bundle directives.	(line 9)
* instruction expansion, CRIS:	CRIS-Expand.	(line 6)
* instruction expansion, MMIX:	MMIX-Expand.	(line 6)
* instruction formats, s390:	s390 Formats.	(line 6)
* instruction marker, s390:	s390 Instruction Marker.	(line 6)
* instruction mnemonics, s390:	s390 Mnemonics.	(line 6)
* instruction naming, i386:	i386-Mnemonics.	(line 9)
* instruction naming, x86-64:	i386-Mnemonics.	(line 9)
* instruction operand modifier, s390:	s390 Operand Modifier.	(line 6)
* instruction operands, s390:	s390 Operands.	(line 6)
* instruction prefixes, i386:	i386-Prefixes.	(line 6)
* instruction set, M680x0:	M68K-opcodes.	(line 6)
* instruction set, M68HC11:	M68HC11-opcodes.	(line 6)
* instruction set, XGATE:	XGATE-opcodes.	(line 5)
* instruction summary, AVR:	AVR Opcodes.	(line 6)
* instruction summary, D10V:	D10V-Opcodes.	(line 6)
* instruction summary, D30V:	D30V-Opcodes.	(line 6)
* instruction summary, H8/300:	H8/300 Opcodes.	(line 6)
* instruction summary, LM32:	LM32 Opcodes.	(line 6)
* instruction summary, SH:	SH Opcodes.	(line 6)
* instruction summary, SH64:	SH64 Opcodes.	(line 6)
* instruction summary, Z8000:	Z8000 Opcodes.	(line 6)
* instruction syntax, s390:	s390 Syntax.	(line 6)
* instructions and directives:	Statements.	(line 20)
* int directive:	Int.	(line 6)
* int directive, H8/300:	H8/300 Directives.	(line 6)
* int directive, i386:	i386-Float.	(line 21)
* int directive, TIC54X:	TIC54X-Directives.	(line 109)
* int directive, x86-64:	i386-Float.	(line 21)
* integer expressions:	Integer Exprs.	(line 6)
* integer, 16-byte:	Octa.	(line 6)
* integer, 8-byte:	Quad.	(line 9)
* integers:	Integers.	(line 6)
* integers, 16-bit:	hword.	(line 6)
* integers, 32-bit:	Int.	(line 6)
* integers, binary:	Integers.	(line 6)
* integers, decimal:	Integers.	(line 12)
* integers, hexadecimal:	Integers.	(line 15)

* integers, octal:	Integers.	(line 9)
* integers, one byte:	Byte.	(line 6)
* intel_syntax pseudo op, i386:	i386-Variations.	(line 6)
* intel_syntax pseudo op, x86-64:	i386-Variations.	(line 6)
* internal assembler sections:	As Sections.	(line 6)
* internal directive:	Internal.	(line 6)
* interrupt link register, ARC:	ARC-Regs.	(line 27)
* Interrupt Vector Base address, ARC:	ARC-Regs.	(line 66)
* invalid input:	Bug Criteria.	(line 14)
* invocation summary:	Overview.	(line 6)
* IP2K architecture options:	IP2K-Opts.	(line 9)
* IP2K architecture options <1>:	IP2K-Opts.	(line 14)
* IP2K line comment character:	IP2K-Chars.	(line 6)
* IP2K line separator:	IP2K-Chars.	(line 14)
* IP2K options:	IP2K-Opts.	(line 6)
* IP2K support:	IP2K-Dependent.	(line 6)
* irp directive:	Irp.	(line 6)
* irpc directive:	Irpc.	(line 6)
* ISA options, SH64:	SH64 Options.	(line 6)
* joining text and data sections:	R.	(line 6)
* jump instructions, i386:	i386-Mnemonics.	(line 58)
* jump instructions, relaxation:	Xtensa Jump Relaxation.	(line 6)
* jump instructions, x86-64:	i386-Mnemonics.	(line 58)
* jump optimization, i386:	i386-Jumps.	(line 6)
* jump optimization, x86-64:	i386-Jumps.	(line 6)
* jump/call operands, i386:	i386-Variations.	(line 15)
* jump/call operands, x86-64:	i386-Variations.	(line 15)
* L16SI instructions, relaxation:	Xtensa Immediate Relaxation.	(line 23)
* L16UI instructions, relaxation:	Xtensa Immediate Relaxation.	(line 23)
* L32I instructions, relaxation:	Xtensa Immediate Relaxation.	(line 23)
* L8UI instructions, relaxation:	Xtensa Immediate Relaxation.	(line 23)
* label (:):	Statements.	(line 31)
* label directive, TIC54X:	TIC54X-Directives.	(line 121)
* labels:	Labels.	(line 6)
* lcomm directive:	Lcomm.	(line 6)
* lcomm directive <1>:	ARC Directives.	(line 9)
* lcomm directive, COFF:	i386-Directives.	(line 6)
* lcommon directive, ARC:	ARC Directives.	(line 24)
* ld:	Object.	(line 15)
* ldouble directive M680x0:	M68K-Float.	(line 17)
* ldouble directive M68HC11:	M68HC11-Float.	(line 17)
* ldouble directive XGATE:	XGATE-Float.	(line 16)
* ldouble directive, TIC54X:	TIC54X-Directives.	(line 62)
* LDR reg,=<expr> pseudo op, AArch64:	AArch64 Opcodes.	(line 9)
* LDR reg,=<label> pseudo op, ARM:	ARM Opcodes.	(line 15)
* leafproc directive, i960:	Directives-i960.	(line 18)
* length directive, TIC54X:	TIC54X-Directives.	(line 125)
* length of symbols:	Symbol Intro.	(line 19)
* level 1 interrupt link register, ARC:	ARC-Regs.	(line 23)
* level 2 interrupt link register, ARC:	ARC-Regs.	(line 31)
* lflags directive (ignored):	Lflags.	(line 6)
* line:	ARC-Chars.	(line 30)
* line comment character:	Comments.	(line 19)
* line comment character, AArch64:	AArch64-Chars.	(line 6)

* line comment character, Alpha:	Alpha-Chars.	(line 6)
* line comment character, ARC:	ARC-Chars.	(line 11)
* line comment character, ARM:	ARM-Chars.	(line 6)
* line comment character, AVR:	AVR-Chars.	(line 6)
* line comment character, CR16:	CR16-Chars.	(line 6)
* line comment character, D10V:	D10V-Chars.	(line 6)
* line comment character, D30V:	D30V-Chars.	(line 6)
* line comment character, Epiphany:	Epiphany-Chars.	(line 6)
* line comment character, H8/300:	H8/300-Chars.	(line 6)
* line comment character, i386:	i386-Chars.	(line 6)
* line comment character, i860:	i860-Chars.	(line 6)
* line comment character, i960:	i960-Chars.	(line 6)
* line comment character, IA-64:	IA-64-Chars.	(line 6)
* line comment character, IP2K:	IP2K-Chars.	(line 6)
* line comment character, LM32:	LM32-Chars.	(line 6)
* line comment character, M32C:	M32C-Chars.	(line 6)
* line comment character, M680x0:	M68K-Chars.	(line 6)
* line comment character, M68HC11:	M68HC11-Syntax.	(line 17)
* line comment character, Meta:	Meta-Chars.	(line 6)
* line comment character, MicroBlaze:	MicroBlaze-Chars.	(line 6)
* line comment character, MIPS:	MIPS-Chars.	(line 6)
* line comment character, MSP 430:	MSP430-Chars.	(line 6)
* line comment character, Nios II:	Nios II Chars.	(line 6)
* line comment character, NS32K:	NS32K-Chars.	(line 6)
* line comment character, PJ:	PJ-Chars.	(line 6)
* line comment character, PowerPC:	PowerPC-Chars.	(line 6)
* line comment character, RL78:	RL78-Chars.	(line 6)
* line comment character, RX:	RX-Chars.	(line 6)
* line comment character, s390:	s390 Characters.	(line 6)
* line comment character, SCORE:	SCORE-Chars.	(line 6)
* line comment character, SH:	SH-Chars.	(line 6)
* line comment character, SH64:	SH64-Chars.	(line 6)
* line comment character, Sparc:	Sparc-Chars.	(line 6)
* line comment character, TIC54X:	TIC54X-Chars.	(line 6)
* line comment character, TIC6X:	TIC6X Syntax.	(line 6)
* line comment character, V850:	V850-Chars.	(line 6)
* line comment character, VAX:	VAX-Chars.	(line 6)
* line comment character, Visium:	Visium Characters.	(line 6)
* line comment character, XGATE:	XGATE-Syntax.	(line 16)
* line comment character, XStormy16:	XStormy16-Chars.	(line 6)
* line comment character, Z80:	Z80-Chars.	(line 6)
* line comment character, Z8000:	Z8000-Chars.	(line 6)
* line comment characters, CRIS:	CRIS-Chars.	(line 6)
* line comment characters, MMIX:	MMIX-Chars.	(line 6)
* line directive:	Line.	(line 6)
* line directive, MSP 430:	MSP430 Directives.	(line 14)
* line numbers, in input files:	Input Files.	(line 35)
* line separator character:	Statements.	(line 6)
* line separator character, Nios II:	Nios II Chars.	(line 6)
* line separator, AArch64:	AArch64-Chars.	(line 10)
* line separator, Alpha:	Alpha-Chars.	(line 11)
* line separator, ARC:	ARC-Chars.	(line 27)
* line separator, ARM:	ARM-Chars.	(line 14)
* line separator, AVR:	AVR-Chars.	(line 14)
* line separator, CR16:	CR16-Chars.	(line 12)
* line separator, Epiphany:	Epiphany-Chars.	(line 14)
* line separator, H8/300:	H8/300-Chars.	(line 8)
* line separator, i386:	i386-Chars.	(line 18)
* line separator, i860:	i860-Chars.	(line 14)

* line separator, i960:	i960-Chars.	(line 14)
* line separator, IA-64:	IA-64-Chars.	(line 8)
* line separator, IP2K:	IP2K-Chars.	(line 14)
* line separator, LM32:	LM32-Chars.	(line 12)
* line separator, M32C:	M32C-Chars.	(line 14)
* line separator, M680x0:	M68K-Chars.	(line 20)
* line separator, M68HC11:	M68HC11-Syntax.	(line 26)
* line separator, Meta:	Meta-Chars.	(line 8)
* line separator, MicroBlaze:	MicroBlaze-Chars.	(line 14)
* line separator, MIPS:	MIPS-Chars.	(line 14)
* line separator, MSP 430:	MSP430-Chars.	(line 14)
* line separator, NS32K:	NS32K-Chars.	(line 18)
* line separator, PJ:	PJ-Chars.	(line 14)
* line separator, PowerPC:	PowerPC-Chars.	(line 18)
* line separator, RL78:	RL78-Chars.	(line 14)
* line separator, RX:	RX-Chars.	(line 14)
* line separator, s390:	s390 Characters.	(line 13)
* line separator, SCORE:	SCORE-Chars.	(line 14)
* line separator, SH:	SH-Chars.	(line 8)
* line separator, SH64:	SH64-Chars.	(line 13)
* line separator, Sparc:	Sparc-Chars.	(line 14)
* line separator, TIC54X:	TIC54X-Chars.	(line 17)
* line separator, TIC6X:	TIC6X Syntax.	(line 13)
* line separator, V850:	V850-Chars.	(line 13)
* line separator, VAX:	VAX-Chars.	(line 14)
* line separator, Visium:	Visium Characters.	(line 14)
* line separator, XGATE:	XGATE-Syntax.	(line 25)
* line separator, XStormy16:	XStormy16-Chars.	(line 14)
* line separator, Z80:	Z80-Chars.	(line 13)
* line separator, Z8000:	Z8000-Chars.	(line 13)
* lines starting with #:	Comments.	(line 33)
* link register, ARC:	ARC-Regs.	(line 35)
* linker:	Object.	(line 15)
* linker, and assembler:	Secs Background.	(line 10)
* linkonce directive:	Linkonce.	(line 6)
* list directive:	List.	(line 6)
* list directive, TIC54X:	TIC54X-Directives.	(line 129)
* listing control, turning off:	Nolist.	(line 6)
* listing control, turning on:	List.	(line 6)
* listing control: new page:	Eject.	(line 6)
* listing control: paper size:	Psize.	(line 6)
* listing control: subtitle:	Sbttl.	(line 6)
* listing control: title line:	Title.	(line 6)
* listings, enabling:	a.	(line 6)
* literal directive:	Literal Directive.	(line 6)
* literal pool entries, s390:	s390 Literal Pool Entries.	(line 6)
* literal_position directive:	Literal Position Directive.	(line 6)
* literal_prefix directive:	Literal Prefix Directive.	(line 6)
* little endian output, MIPS:	Overview.	(line 809)
* little endian output, PJ:	Overview.	(line 716)
* little-endian output, MIPS:	MIPS Options.	(line 13)
* little-endian output, TIC6X:	TIC6X Options.	(line 46)
* LM32 line comment character:	LM32-Chars.	(line 6)
* LM32 line separator:	LM32-Chars.	(line 12)
* LM32 modifiers:	LM32-Modifiers.	(line 6)
* LM32 opcode summary:	LM32 Opcodes.	(line 6)

* LM32 options (none):	LM32 Options.	(line 6)
* LM32 register names:	LM32-Regs.	(line 6)
* LM32 support:	LM32-Dependent.	(line 6)
* ln directive:	Ln.	(line 6)
* lo directive, Nios II:	Nios II Relocations.	(line 23)
* lo pseudo-op, V850:	V850 Opcodes.	(line 22)
* loc directive:	Loc.	(line 6)
* local common symbols:	Lcomm.	(line 6)
* local directive:	Local.	(line 6)
* local labels:	Symbol Names.	(line 43)
* local symbol names:	Symbol Names.	(line 30)
* local symbols, retaining in output:	L.	(line 6)
* location counter:	Dot.	(line 6)
* location counter, advancing:	Org.	(line 6)
* location counter, Z80:	Z80-Chars.	(line 15)
* loc_mark_labels directive:	Loc_mark_labels.	(line 6)
* logical file name:	File.	(line 13)
* logical line number:	Line.	(line 6)
* logical line numbers:	Comments.	(line 33)
* long directive:	Long.	(line 6)
* long directive, i386:	i386-Float.	(line 21)
* long directive, TIC54X:	TIC54X-Directives.	(line 133)
* long directive, x86-64:	i386-Float.	(line 21)
* longcall pseudo-op, V850:	V850 Opcodes.	(line 122)
* longcalls directive:	Longcalls Directive.	(line 6)
* longjump pseudo-op, V850:	V850 Opcodes.	(line 128)
* loop counter, ARC:	ARC-Regs.	(line 41)
* loop directive, TIC54X:	TIC54X-Directives.	(line 141)
* LOOP instructions, alignment:	Xtensa Automatic Alignment.	(line 6)
* low directive, M32R:	M32R-Directives.	(line 9)
* lp register, V850:	V850-Regs.	(line 68)
* lval:	Z8000 Directives.	(line 27)
* LWP, i386:	i386-LWP.	(line 6)
* LWP, x86-64:	i386-LWP.	(line 6)
* M16C architecture option:	M32C-Opts.	(line 12)
* M32C architecture option:	M32C-Opts.	(line 9)
* M32C line comment character:	M32C-Chars.	(line 6)
* M32C line separator:	M32C-Chars.	(line 14)
* M32C modifiers:	M32C-Modifiers.	(line 6)
* M32C options:	M32C-Opts.	(line 6)
* M32C support:	M32C-Dependent.	(line 6)
* M32R architecture options:	M32R-Opts.	(line 9)
* M32R architecture options <1>:	M32R-Opts.	(line 17)
* M32R architecture options <2>:	M32R-Opts.	(line 21)
* M32R directives:	M32R-Directives.	(line 6)
* M32R options:	M32R-Opts.	(line 6)
* M32R support:	M32R-Dependent.	(line 6)
* M32R warnings:	M32R-Warnings.	(line 6)
* M680x0 addressing modes:	M68K-Syntax.	(line 21)
* M680x0 architecture options:	M68K-Opts.	(line 99)
* M680x0 branch improvement:	M68K-Branch.	(line 6)
* M680x0 directives:	M68K-Directives.	(line 6)
* M680x0 floating point:	M68K-Float.	(line 6)
* M680x0 immediate character:	M68K-Chars.	(line 13)
* M680x0 line comment character:	M68K-Chars.	(line 6)
* M680x0 line separator:	M68K-Chars.	(line 20)

```

* M680x0 opcodes: M68K-opcodes. (line 6)
* M680x0 options: M68K-Opts. (line 6)
* M680x0 pseudo-opcodes: M68K-Branch. (line 6)
* M680x0 size modifiers: M68K-Syntax. (line 8)
* M680x0 support: M68K-Dependent. (line 6)
* M680x0 syntax: M68K-Syntax. (line 8)
* M68HC11 addressing modes: M68HC11-Syntax. (line 29)
* M68HC11 and M68HC12 support: M68HC11-Dependent. (line 6)
* M68HC11 assembler directive .far: M68HC11-Directives. (line 20)
* M68HC11 assembler directive .interrupt: M68HC11-Directives. (line 26)
* M68HC11 assembler directive .mode: M68HC11-Directives. (line 16)
* M68HC11 assembler directive .relax: M68HC11-Directives. (line 10)
* M68HC11 assembler directive .xrefb: M68HC11-Directives. (line 31)
* M68HC11 assembler directives: M68HC11-Directives. (line 6)
* M68HC11 branch improvement: M68HC11-Branch. (line 6)
* M68HC11 floating point: M68HC11-Float. (line 6)
* M68HC11 line comment character: M68HC11-Syntax. (line 17)
* M68HC11 line separator: M68HC11-Syntax. (line 26)
* M68HC11 modifiers: M68HC11-Modifiers. (line 6)
* M68HC11 opcodes: M68HC11-opcodes. (line 6)
* M68HC11 options: M68HC11-Opts. (line 6)
* M68HC11 pseudo-opcodes: M68HC11-Branch. (line 6)
* M68HC11 syntax: M68HC11-Syntax. (line 6)
* M68HC12 assembler directives: M68HC11-Directives. (line 6)
* mA6 command line option, ARC: ARC Options. (line 14)
* mA7 command line option, ARC: ARC Options. (line 39)
* machine dependencies: Machine Dependencies. (line 6)
* machine directives, AArch64: AArch64 Directives. (line 6)
* machine directives, ARC: ARC Directives. (line 6)
* machine directives, ARM: ARM Directives. (line 6)
* machine directives, H8/300 (none): H8/300 Directives. (line 6)
* machine directives, i860: Directives-i860. (line 6)
* machine directives, i960: Directives-i960. (line 6)
* machine directives, MSP 430: MSP430 Directives. (line 6)
* machine directives, Nios II: Nios II Directives. (line 6)
* machine directives, SH: SH Directives. (line 6)
* machine directives, SH64: SH64 Directives. (line 9)
* machine directives, SPARC: Sparc-Directives. (line 6)
* machine directives, TIC54X: TIC54X-Directives. (line 6)
* machine directives, TIC6X: TIC6X Directives. (line 6)
* machine directives, TILE-Gx: TILE-Gx Directives. (line 6)
* machine directives, TILEPro: TILEPro Directives. (line 6)
* machine directives, V850: V850 Directives. (line 6)
* machine directives, VAX: VAX-directives. (line 6)
* machine directives, x86: i386-Directives. (line 6)
* machine directives, XStormy16: XStormy16 Directives. (line 6)
* machine independent directives: Pseudo Ops. (line 6)
* machine instructions (not covered): Manual. (line 14)
* machine relocations, Nios II: Nios II Relocations. (line 6)
* machine-independent syntax: Syntax. (line 6)
* macro directive: Macro. (line 28)
* macro directive, TIC54X: TIC54X-Directives. (line 151)
* macros: Macro. (line 6)
* macros, count executed: Macro. (line 142)
* Macros, MSP 430: MSP430-Macros. (line 6)

```

```

* macros, TIC54X: TIC54X-Macros. (line 6)
* make rules: MD. (line 6)
* manual, structure and purpose: Manual. (line 6)
* marc600 command line option, ARC: ARC Options. (line 14)
* mARC601 command line option, ARC: ARC Options. (line 27)
* mARC700 command line option, ARC: ARC Options. (line 39)
* march command line option, Nios II: Nios II Options. (line 28)
* math builtins, TIC54X: TIC54X-Builtins. (line 6)
* Maximum number of continuation lines: listing. (line 34)
* mEM command line option, ARC: ARC Options. (line 42)
* memory references, i386: i386-Memory. (line 6)
* memory references, x86-64: i386-Memory. (line 6)
* memory-mapped registers, TIC54X: TIC54X-MMRegs. (line 6)
* merging text and data sections: R. (line 6)
* messages from assembler: Errors. (line 6)
* Meta architectures: Meta Options. (line 6)
* Meta line comment character: Meta-Chars. (line 6)
* Meta line separator: Meta-Chars. (line 8)
* Meta options: Meta Options. (line 6)
* Meta registers: Meta-Regs. (line 6)
* Meta support: Meta-Dependent. (line 6)
* mHS command line option, ARC: ARC Options. (line 64)
* MicroBlaze architectures: MicroBlaze-Dependent.
                                (line 6)
* MicroBlaze directives: MicroBlaze Directives.
                                (line 6)
* MicroBlaze line comment character: MicroBlaze-Chars. (line 6)
* MicroBlaze line separator: MicroBlaze-Chars. (line 14)
* MicroBlaze support: MicroBlaze-Dependent.
                                (line 12)
* minus, permitted arguments: Infix Ops. (line 50)
* MIPS 32-bit microMIPS instruction generation override: MIPS assembly options.
                                (line 18)
* MIPS architecture options: MIPS Options. (line 29)
* MIPS big-endian output: MIPS Options. (line 13)
* MIPS CPU override: MIPS ISA. (line 18)
* MIPS directives to override command line options: MIPS assembly options.
                                (line 6)
* MIPS DSP Release 1 instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                (line 21)
* MIPS DSP Release 2 instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                (line 26)
* MIPS DSP Release 3 instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                (line 31)
* MIPS endianness: Overview. (line 806)
* MIPS eXtended Physical Address (XPA) instruction generation override: MIPS ASE
Instruction Generation Overrides.
                                (line 57)
* MIPS IEEE 754 NaN data encoding selection: MIPS NaN Encodings.
                                (line 6)
* MIPS ISA: Overview. (line 812)
* MIPS ISA override: MIPS ISA. (line 6)
* MIPS line comment character: MIPS-Chars. (line 6)
* MIPS line separator: MIPS-Chars. (line 14)
* MIPS little-endian output: MIPS Options. (line 13)
* MIPS MCU instruction generation override: MIPS ASE Instruction Generation

```

```

* MIPS MCU instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                                    (line 42)
* MIPS MDMX instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                                    (line 16)
* MIPS MIPS-3D instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                                    (line 6)
* MIPS MT instruction generation override: MIPS ASE Instruction Generation Overrides.
                                                    (line 37)
* MIPS option stack:                MIPS Option Stack. (line 6)
* MIPS processor:                   MIPS-Dependent.   (line 6)
* MIPS SIMD Architecture instruction generation override: MIPS ASE Instruction
Generation Overrides.
                                                    (line 47)
* MIT:                               M68K-Syntax.    (line 6)
* mlib directive, TIC54X:            TIC54X-Directives. (line 157)
* mlist directive, TIC54X:          TIC54X-Directives. (line 162)
* MMIX assembler directive BSPEC:   MMIX-Pseudos.    (line 137)
* MMIX assembler directive BYTE:    MMIX-Pseudos.    (line 101)
* MMIX assembler directive ESPEC:   MMIX-Pseudos.    (line 137)
* MMIX assembler directive GREG:    MMIX-Pseudos.    (line 53)
* MMIX assembler directive IS:      MMIX-Pseudos.    (line 44)
* MMIX assembler directive LOC:     MMIX-Pseudos.    (line 7)
* MMIX assembler directive LOCAL:   MMIX-Pseudos.    (line 29)
* MMIX assembler directive OCTA:    MMIX-Pseudos.    (line 113)
* MMIX assembler directive PREFIX:  MMIX-Pseudos.    (line 125)
* MMIX assembler directive TETRA:   MMIX-Pseudos.    (line 113)
* MMIX assembler directive WYDE:    MMIX-Pseudos.    (line 113)
* MMIX assembler directives:        MMIX-Pseudos.    (line 6)
* MMIX line comment characters:     MMIX-Chars.      (line 6)
* MMIX options:                     MMIX-Opts.       (line 6)
* MMIX pseudo-op BSPEC:             MMIX-Pseudos.    (line 137)
* MMIX pseudo-op BYTE:             MMIX-Pseudos.    (line 101)
* MMIX pseudo-op ESPEC:            MMIX-Pseudos.    (line 137)
* MMIX pseudo-op GREG:             MMIX-Pseudos.    (line 53)
* MMIX pseudo-op IS:               MMIX-Pseudos.    (line 44)
* MMIX pseudo-op LOC:              MMIX-Pseudos.    (line 7)
* MMIX pseudo-op LOCAL:            MMIX-Pseudos.    (line 29)
* MMIX pseudo-op OCTA:             MMIX-Pseudos.    (line 113)
* MMIX pseudo-op PREFIX:           MMIX-Pseudos.    (line 125)
* MMIX pseudo-op TETRA:            MMIX-Pseudos.    (line 113)
* MMIX pseudo-op WYDE:             MMIX-Pseudos.    (line 113)
* MMIX pseudo-ops:                 MMIX-Pseudos.    (line 6)
* MMIX register names:             MMIX-Regs.       (line 6)
* MMIX support:                     MMIX-Dependent.  (line 6)
* mmixal differences:              MMIX-mmixal.     (line 6)
* mmregs directive, TIC54X:        TIC54X-Directives. (line 167)
* mmsg directive, TIC54X:          TIC54X-Directives. (line 75)
* MMX, i386:                       i386-SIMD.       (line 6)
* MMX, x86-64:                     i386-SIMD.       (line 6)
* mnemonic compatibility, i386:     i386-Mnemonics.  (line 64)
* mnemonic suffixes, i386:         i386-Variations. (line 28)
* mnemonic suffixes, x86-64:       i386-Variations. (line 28)
* mnemonics for opcodes, VAX:      VAX-opcodes.     (line 6)
* mnemonics, AVR:                  AVR Opcodes.     (line 6)
* mnemonics, D10V:                 D10V-Opcodes.    (line 6)
* mnemonics, D30V:                 D30V-Opcodes.    (line 6)

```

```

* mnemonics, H8/300:      H8/300 Opcodes.      (line 6)
* mnemonics, LM32:       LM32 Opcodes.        (line 6)
* mnemonics, SH:         SH Opcodes.            (line 6)
* mnemonics, SH64:       SH64 Opcodes.          (line 6)
* mnemonics, Z8000:      Z8000 Opcodes.         (line 6)
* mnolist directive, TIC54X: TIC54X-Directives. (line 162)
* mnps400 command line option, ARC: ARC Options.      (line 79)
* modifiers, M32C:       M32C-Modifiers.        (line 6)
* Motorola syntax for the 680x0: M68K-Moto-Syntax. (line 6)
* MOVI instructions, relaxation: Xtensa Immediate Relaxation.
                                     (line 12)
* MOVN, MOVZ and MOVK group relocations, AArch64: AArch64-Relocations.
                                     (line 6)
* MOVW and MOVT relocations, ARM: ARM-Relocations. (line 21)
* MRI compatibility mode: M. (line 6)
* mri directive:        MRI. (line 6)
* MRI mode, temporarily: MRI. (line 6)
* MSP 430 floating point (IEEE): MSP430 Floating Point.
                                     (line 6)
* MSP 430 identifiers:   MSP430-Chars. (line 17)
* MSP 430 line comment character: MSP430-Chars. (line 6)
* MSP 430 line separator: MSP430-Chars. (line 14)
* MSP 430 machine directives: MSP430 Directives. (line 6)
* MSP 430 macros:       MSP430-Macros. (line 6)
* MSP 430 opcodes:      MSP430 Opcodes. (line 6)
* MSP 430 options (none): MSP430 Options. (line 6)
* MSP 430 profiling capability: MSP430 Profiling Capability.
                                     (line 6)
* MSP 430 register names: MSP430-Regs. (line 6)
* MSP 430 support:      MSP430-Dependent. (line 6)
* MSP430 Assembler Extensions: MSP430-Ext. (line 6)
* mul instruction, i386: i386-Notes. (line 6)
* mul instruction, x86-64: i386-Notes. (line 6)
* N32K support:        NS32K-Dependent. (line 6)
* name:                Z8000 Directives. (line 18)
* named section:       Section. (line 6)
* named sections:     Ld Sections. (line 8)
* names, symbol:      Symbol Names. (line 6)
* naming object file: o. (line 6)
* NDS32 options:      NDS32 Options. (line 6)
* NDS32 processor:    NDS32-Dependent. (line 6)
* new page, in listings: Eject. (line 6)
* newblock directive, TIC54X: TIC54X-Directives. (line 173)
* newline (\n):       Strings. (line 21)
* newline, required at file end: Statements. (line 14)
* Nios II line comment character: Nios II Chars. (line 6)
* Nios II line separator character: Nios II Chars. (line 6)
* Nios II machine directives: Nios II Directives. (line 6)
* Nios II machine relocations: Nios II Relocations.
                                     (line 6)
* Nios II opcodes:     Nios II Opcodes. (line 6)
* Nios II options:     Nios II Options. (line 6)
* Nios II support:     NiosII-Dependent. (line 6)
* Nios support:        NiosII-Dependent. (line 6)
* no-absolute-literals directive: Absolute Literals Directive.
                                     (line 6)
* no-longcalls directive: Longcalls Directive.
                                     (line 6)
* no-relax command line option, Nios II: Nios II Options. (line 19)

```


* no-schedule directive:	Schedule Directive.	(line 6)
* no-transform directive:	Transform Directive.	(line 6)
* nolist directive:	Nolist.	(line 6)
* nolist directive, TIC54X:	TIC54X-Directives.	(line 129)
* NOP pseudo op, ARM:	ARM Opcodes.	(line 9)
* notes for Alpha:	Alpha Notes.	(line 6)
* NS32K line comment character:	NS32K-Chars.	(line 6)
* NS32K line separator:	NS32K-Chars.	(line 18)
* null-terminated strings:	Asciz.	(line 6)
* number constants:	Numbers.	(line 6)
* number of macros executed:	Macro.	(line 142)
* numbered subsections:	Sub-Sections.	(line 6)
* numbers, 16-bit:	hword.	(line 6)
* numeric values:	Expressions.	(line 6)
* nword directive, SPARC:	Sparc-Directives.	(line 20)
* object attributes:	Object Attributes.	(line 6)
* object file:	Object.	(line 6)
* object file format:	Object Formats.	(line 6)
* object file name:	o.	(line 6)
* object file, after errors:	Z.	(line 6)
* obsolescent directives:	Deprecated.	(line 6)
* octa directive:	Octa.	(line 6)
* octal character code (\DDD):	Strings.	(line 30)
* octal integers:	Integers.	(line 9)
* offset directive:	Offset.	(line 6)
* offset directive, V850:	V850 Directives.	(line 6)
* opcode mnemonics, VAX:	VAX-opcodes.	(line 6)
* opcode names, TILE-Gx:	TILE-Gx Opcodes.	(line 6)
* opcode names, TILEPro:	TILEPro Opcodes.	(line 6)
* opcode names, Xtensa:	Xtensa Opcodes.	(line 6)
* opcode summary, AVR:	AVR Opcodes.	(line 6)
* opcode summary, D10V:	D10V-Opcodes.	(line 6)
* opcode summary, D30V:	D30V-Opcodes.	(line 6)
* opcode summary, H8/300:	H8/300 Opcodes.	(line 6)
* opcode summary, LM32:	LM32 Opcodes.	(line 6)
* opcode summary, SH:	SH Opcodes.	(line 6)
* opcode summary, SH64:	SH64 Opcodes.	(line 6)
* opcode summary, Z8000:	Z8000 Opcodes.	(line 6)
* opcodes for AArch64:	AArch64 Opcodes.	(line 6)
* opcodes for ARC:	ARC Opcodes.	(line 6)
* opcodes for ARM:	ARM Opcodes.	(line 6)
* opcodes for MSP 430:	MSP430 Opcodes.	(line 6)
* opcodes for Nios II:	Nios II Opcodes.	(line 6)
* opcodes for V850:	V850 Opcodes.	(line 6)
* opcodes, i860:	Opcodes for i860.	(line 6)
* opcodes, i960:	Opcodes for i960.	(line 6)
* opcodes, M680x0:	M68K-opcodes.	(line 6)
* opcodes, M68HC11:	M68HC11-opcodes.	(line 6)
* operand delimiters, i386:	i386-Variations.	(line 15)
* operand delimiters, x86-64:	i386-Variations.	(line 15)
* operand notation, VAX:	VAX-operands.	(line 6)
* operands in expressions:	Arguments.	(line 6)
* operator precedence:	Infix Ops.	(line 11)
* operators, in expressions:	Operators.	(line 6)
* operators, permitted arguments:	Infix Ops.	(line 6)
* optimization, D10V:	Overview.	(line 579)
* optimization, D30V:	Overview.	(line 584)
* optimizations:	Xtensa Optimizations.	

		(line	6)
* option directive, TIC54X:	TIC54X-Directives.	(line	177)
* option summary:	Overview.	(line	6)
* options for AArch64 (none):	AArch64 Options.	(line	6)
* options for Alpha:	Alpha Options.	(line	6)
* options for ARC:	ARC Options.	(line	6)
* options for ARM (none):	ARM Options.	(line	6)
* options for AVR (none):	AVR Options.	(line	6)
* options for Blackfin (none):	Blackfin Options.	(line	6)
* options for i386:	i386-Options.	(line	6)
* options for IA-64:	IA-64 Options.	(line	6)
* options for LM32 (none):	LM32 Options.	(line	6)
* options for Meta:	Meta Options.	(line	6)
* options for MSP430 (none):	MSP430 Options.	(line	6)
* options for NDS32:	NDS32 Options.	(line	6)
* options for Nios II:	Nios II Options.	(line	6)
* options for PDP-11:	PDP-11-Options.	(line	6)
* options for PowerPC:	PowerPC-Opts.	(line	6)
* options for s390:	s390 Options.	(line	6)
* options for SCORE:	SCORE-Opts.	(line	6)
* options for SPARC:	Sparc-Opts.	(line	6)
* options for TIC6X:	TIC6X Options.	(line	6)
* options for V850 (none):	V850 Options.	(line	6)
* options for VAX/VMS:	VAX-Opts.	(line	42)
* options for Visium:	Visium Options.	(line	6)
* options for x86-64:	i386-Options.	(line	6)
* options for Z80:	Z80 Options.	(line	6)
* options, all versions of assembler:	Invoking.	(line	6)
* options, command line:	Command Line.	(line	13)
* options, CRIS:	CRIS-Opts.	(line	6)
* options, D10V:	D10V-Opts.	(line	6)
* options, D30V:	D30V-Opts.	(line	6)
* options, Epiphany:	Epiphany Options.	(line	6)
* options, H8/300:	H8/300 Options.	(line	6)
* options, i960:	Options-i960.	(line	6)
* options, IP2K:	IP2K-Opts.	(line	6)
* options, M32C:	M32C-Opts.	(line	6)
* options, M32R:	M32R-Opts.	(line	6)
* options, M680x0:	M68K-Opts.	(line	6)
* options, M68HC11:	M68HC11-Opts.	(line	6)
* options, MMIX:	MMIX-Opts.	(line	6)
* options, PJ:	PJ Options.	(line	6)
* options, RL78:	RL78-Opts.	(line	6)
* options, RX:	RX-Opts.	(line	6)
* options, SH:	SH Options.	(line	6)
* options, SH64:	SH64 Options.	(line	6)
* options, TIC54X:	TIC54X-Opts.	(line	6)
* options, XGATE:	XGATE-Opts.	(line	6)
* options, Z8000:	Z8000 Options.	(line	6)
* org directive:	Org.	(line	6)
* other attribute, of a.out symbol:	Symbol Other.	(line	6)
* output file:	Object.	(line	6)
* output section padding:	no-pad-sections.	(line	6)
* p2align directive:	P2align.	(line	6)
* p2alignl directive:	P2align.	(line	28)
* p2alignw directive:	P2align.	(line	28)
* padding the location counter:	Align.	(line	6)
* padding the location counter given a power of two: P2align.		(line	6)

```

* padding the location counter given number of bytes: Balign.
                                                                    (line 6)
* page, in listings: Eject. (line 6)
* paper size, for listings: Psize. (line 6)
* paths for .include: I. (line 6)
* patterns, writing in memory: Fill. (line 6)
* PDP-11 comments: PDP-11-Syntax. (line 16)
* PDP-11 floating-point register syntax: PDP-11-Syntax. (line 13)
* PDP-11 general-purpose register syntax: PDP-11-Syntax. (line 10)
* PDP-11 instruction naming: PDP-11-Mnemonics. (line 6)
* PDP-11 line separator: PDP-11-Syntax. (line 19)
* PDP-11 support: PDP-11-Dependent. (line 6)
* PDP-11 syntax: PDP-11-Syntax. (line 6)
* PIC code generation for ARM: ARM Options. (line 187)
* PIC code generation for M32R: M32R-Opts. (line 42)
* PIC selection, MIPS: MIPS Options. (line 21)
* PJ endianness: Overview. (line 713)
* PJ line comment character: PJ-Chars. (line 6)
* PJ line separator: PJ-Chars. (line 14)
* PJ options: PJ Options. (line 6)
* PJ support: PJ-Dependent. (line 6)
* plus, permitted arguments: Infix Ops. (line 45)
* popsection directive: PopSection. (line 6)
* Position-independent code, CRIS: CRIS-Opts. (line 27)
* Position-independent code, symbols in, CRIS: CRIS-Pic. (line 6)
* PowerPC architectures: PowerPC-Opts. (line 6)
* PowerPC directives: PowerPC-Pseudo. (line 6)
* PowerPC line comment character: PowerPC-Chars. (line 6)
* PowerPC line separator: PowerPC-Chars. (line 18)
* PowerPC options: PowerPC-Opts. (line 6)
* PowerPC support: PPC-Dependent. (line 6)
* precedence of operators: Infix Ops. (line 11)
* precision, floating point: Flonums. (line 6)
* prefix operators: Prefix Ops. (line 6)
* prefixes, i386: i386-Prefixes. (line 6)
* preprocessing: Preprocessing. (line 6)
* preprocessing, turning on and off: Preprocessing. (line 26)
* previous directive: Previous. (line 6)
* primary attributes, COFF symbols: COFF Symbols. (line 13)
* print directive: Print. (line 6)
* proc directive, SPARC: Sparc-Directives. (line 25)
* Processor Identification register, ARC: ARC-Regs. (line 51)
* profiler directive, MSP 430: MSP430 Directives. (line 26)
* profiling capability for MSP 430: MSP430 Profiling Capability.
                                                                    (line 6)
* Program Counter, ARC: ARC-Regs. (line 54)
* protected directive: Protected. (line 6)
* pseudo-op .arch, CRIS: CRIS-Pseudos. (line 50)
* pseudo-op .dword, CRIS: CRIS-Pseudos. (line 12)
* pseudo-op .syntax, CRIS: CRIS-Pseudos. (line 18)
* pseudo-op BSPEC, MMIX: MMIX-Pseudos. (line 137)
* pseudo-op BYTE, MMIX: MMIX-Pseudos. (line 101)
* pseudo-op ESPEC, MMIX: MMIX-Pseudos. (line 137)
* pseudo-op GREG, MMIX: MMIX-Pseudos. (line 53)
* pseudo-op IS, MMIX: MMIX-Pseudos. (line 44)
* pseudo-op LOC, MMIX: MMIX-Pseudos. (line 7)
* pseudo-op LOCAL, MMIX: MMIX-Pseudos. (line 29)
* pseudo-op OCTA, MMIX: MMIX-Pseudos. (line 113)
* pseudo-op PREFIX, MMIX: MMIX-Pseudos. (line 125)

```

```

* pseudo-op TETRA, MMIX: MMIX-Pseudos. (line 113)
* pseudo-op WYDE, MMIX: MMIX-Pseudos. (line 113)
* pseudo-opcodes for XStormy16: XStormy16 Opcodes. (line 6)
* pseudo-opcodes, M680x0: M68K-Branch. (line 6)
* pseudo-opcodes, M68HC11: M68HC11-Branch. (line 6)
* pseudo-ops for branch, VAX: VAX-branch. (line 6)
* pseudo-ops, CRIS: CRIS-Pseudos. (line 6)
* pseudo-ops, machine independent: Pseudo Ops. (line 6)
* pseudo-ops, MMIX: MMIX-Pseudos. (line 6)
* psize directive: Psize. (line 6)
* PSR bits: IA-64-Bits. (line 6)
* pstring directive, TIC54X: TIC54X-Directives. (line 206)
* psw register, V850: V850-Regs. (line 80)
* purgem directive: Purgem. (line 6)
* purpose of GNU assembler: GNU Assembler. (line 12)
* pushsection directive: PushSection. (line 6)
* quad directive: Quad. (line 6)
* quad directive, i386: i386-Float. (line 21)
* quad directive, x86-64: i386-Float. (line 21)
* real-mode code, i386: i386-16bit. (line 6)
* ref directive, TIC54X: TIC54X-Directives. (line 101)
* refsym directive, MSP 430: MSP430 Directives. (line 30)
* register directive, SPARC: Sparc-Directives. (line 29)
* register name prefix character, ARC: ARC-Chars. (line 7)
* register names, AArch64: AArch64-Regs. (line 6)
* register names, Alpha: Alpha-Regs. (line 6)
* register names, ARC: ARC-Regs. (line 6)
* register names, ARM: ARM-Regs. (line 6)
* register names, AVR: AVR-Regs. (line 6)
* register names, CRIS: CRIS-Regs. (line 6)
* register names, H8/300: H8/300-Regs. (line 6)
* register names, IA-64: IA-64-Regs. (line 6)
* register names, LM32: LM32-Regs. (line 6)
* register names, MMIX: MMIX-Regs. (line 6)
* register names, MSP 430: MSP430-Regs. (line 6)
* register names, Sparc: Sparc-Regs. (line 6)
* register names, TILE-Gx: TILE-Gx Registers. (line 6)
* register names, TILEPro: TILEPro Registers. (line 6)
* register names, V850: V850-Regs. (line 6)
* register names, VAX: VAX-operands. (line 17)
* register names, Visium: Visium Registers. (line 6)
* register names, Xtensa: Xtensa Registers. (line 6)
* register names, Z80: Z80-Regs. (line 6)
* register naming, s390: s390 Register. (line 6)
* register operands, i386: i386-Variations. (line 15)
* register operands, x86-64: i386-Variations. (line 15)
* registers, D10V: D10V-Regs. (line 6)
* registers, D30V: D30V-Regs. (line 6)
* registers, i386: i386-Regs. (line 6)
* registers, Meta: Meta-Regs. (line 6)
* registers, SH: SH-Regs. (line 6)
* registers, SH64: SH64-Regs. (line 6)
* registers, TIC54X memory-mapped: TIC54X-MMRegs. (line 6)
* registers, x86-64: i386-Regs. (line 6)
* registers, Z8000: Z8000-Regs. (line 6)
* relax-all command line option, Nios II: Nios II Options. (line 13)
* relax-section command line option, Nios II: Nios II Options. (line 6)
* relaxation: Xtensa Relaxation. (line 6)

```

```

* relaxation of ADDI instructions:      Xtensa Immediate Relaxation.
                                         (line 43)
* relaxation of branch instructions:   Xtensa Branch Relaxation.
                                         (line 6)
* relaxation of call instructions:     Xtensa Call Relaxation.
                                         (line 6)
* relaxation of immediate fields:     Xtensa Immediate Relaxation.
                                         (line 6)
* relaxation of jump instructions:     Xtensa Jump Relaxation.
                                         (line 6)
* relaxation of L16SI instructions:    Xtensa Immediate Relaxation.
                                         (line 23)
* relaxation of L16UI instructions:    Xtensa Immediate Relaxation.
                                         (line 23)
* relaxation of L32I instructions:    Xtensa Immediate Relaxation.
                                         (line 23)
* relaxation of L8UI instructions:     Xtensa Immediate Relaxation.
                                         (line 23)
* relaxation of MOVI instructions:     Xtensa Immediate Relaxation.
                                         (line 12)
* reloc directive:                    Reloc. (line 6)
* relocation:                          Sections. (line 6)
* relocation example:                  Ld Sections. (line 40)
* relocations, AArch64:                AArch64-Relocations.
                                         (line 6)
* relocations, Alpha:                  Alpha-Relocs. (line 6)
* relocations, Sparc:                  Sparc-Relocs. (line 6)
* repeat prefixes, i386:               i386-Prefixes. (line 44)
* reporting bugs in assembler:         Reporting Bugs. (line 6)
* rept directive:                      Rept. (line 6)
* reserve directive, SPARC:            Sparc-Directives. (line 39)
* return instructions, i386:            i386-Variations. (line 40)
* return instructions, x86-64:         i386-Variations. (line 40)
* REX prefixes, i386:                  i386-Prefixes. (line 46)
* RISC-V support:                      RISC-V-Dependent. (line 6)
* RL78 assembler directives:           RL78-Directives. (line 6)
* RL78 line comment character:         RL78-Chars. (line 6)
* RL78 line separator:                RL78-Chars. (line 14)
* RL78 modifiers:                     RL78-Modifiers. (line 6)
* RL78 options:                       RL78-Opts. (line 6)
* RL78 support:                       RL78-Dependent. (line 6)
* rsect:                               Z8000 Directives. (line 52)
* RX assembler directive .3byte:       RX-Directives. (line 9)
* RX assembler directive .fetchalign: RX-Directives. (line 13)
* RX assembler directives:             RX-Directives. (line 6)
* RX floating point:                   RX-Float. (line 6)
* RX line comment character:           RX-Chars. (line 6)
* RX line separator:                   RX-Chars. (line 14)
* RX modifiers:                        RX-Modifiers. (line 6)
* RX options:                          RX-Opts. (line 6)
* RX support:                          RX-Dependent. (line 6)
* s390 floating point:                 s390 Floating Point.
                                         (line 6)
* s390 instruction aliases:            s390 Aliases. (line 6)
* s390 instruction formats:            s390 Formats. (line 6)
* s390 instruction marker:             s390 Instruction Marker.
                                         (line 6)
* s390 instruction mnemonics:          s390 Mnemonics. (line 6)
* s390 instruction operand modifier:   s390 Operand Modifier.

```

		(line	6)
* s390 instruction operands:	s390 Operands.	(line	6)
* s390 instruction syntax:	s390 Syntax.	(line	6)
* s390 line comment character:	s390 Characters.	(line	6)
* s390 line separator:	s390 Characters.	(line	13)
* s390 literal pool entries:	s390 Literal Pool Entries.	(line	6)
* s390 options:	s390 Options.	(line	6)
* s390 register naming:	s390 Register.	(line	6)
* s390 support:	S/390-Dependent.	(line	6)
* Saved User Stack Pointer, ARC:	ARC-Regs.	(line	73)
* sblock directive, TIC54X:	TIC54X-Directives.	(line	180)
* sbttl directive:	Sbttl.	(line	6)
* schedule directive:	Schedule Directive.	(line	6)
* scl directive:	Scl.	(line	6)
* SCORE architectures:	SCORE-Opts.	(line	6)
* SCORE directives:	SCORE-Pseudo.	(line	6)
* SCORE line comment character:	SCORE-Chars.	(line	6)
* SCORE line separator:	SCORE-Chars.	(line	14)
* SCORE options:	SCORE-Opts.	(line	6)
* SCORE processor:	SCORE-Dependent.	(line	6)
* sdaoff pseudo-op, V850:	V850 Opcodes.	(line	65)
* search path for .include:	I.	(line	6)
* sect directive, TIC54X:	TIC54X-Directives.	(line	186)
* section directive (COFF version):	Section.	(line	16)
* section directive (ELF version):	Section.	(line	67)
* section directive, V850:	V850 Directives.	(line	9)
* section name substitution:	Section.	(line	71)
* section override prefixes, i386:	i386-Prefixes.	(line	23)
* Section Stack:	PopSection.	(line	6)
* Section Stack <1>:	Previous.	(line	6)
* Section Stack <2>:	PushSection.	(line	6)
* Section Stack <3>:	Section.	(line	62)
* Section Stack <4>:	SubSection.	(line	6)
* section-relative addressing:	Secs Background.	(line	65)
* sections:	Sections.	(line	6)
* sections in messages, internal:	As Sections.	(line	6)
* sections, i386:	i386-Variations.	(line	46)
* sections, named:	Ld Sections.	(line	8)
* sections, x86-64:	i386-Variations.	(line	46)
* seg directive, SPARC:	Sparc-Directives.	(line	44)
* segm:	Z8000 Directives.	(line	10)
* set at directive, Nios II:	Nios II Directives.	(line	35)
* set break directive, Nios II:	Nios II Directives.	(line	43)
* set directive:	Set.	(line	6)
* set directive, Nios II:	Nios II Directives.	(line	57)
* set directive, TIC54X:	TIC54X-Directives.	(line	189)
* set noat directive, Nios II:	Nios II Directives.	(line	31)
* set nobreak directive, Nios II:	Nios II Directives.	(line	39)
* set norelax directive, Nios II:	Nios II Directives.	(line	46)
* set relaxall directive, Nios II:	Nios II Directives.	(line	53)
* set relaxsection directive, Nios II:	Nios II Directives.	(line	49)
* SH addressing modes:	SH-Addressing.	(line	6)
* SH floating point (IEEE):	SH Floating Point.	(line	6)
* SH line comment character:	SH-Chars.	(line	6)
* SH line separator:	SH-Chars.	(line	8)
* SH machine directives:	SH Directives.	(line	6)
* SH opcode summary:	SH Opcodes.	(line	6)
* SH options:	SH Options.	(line	6)

* SH registers:	SH-Regs.	(line 6)
* SH support:	SH-Dependent.	(line 6)
* SH64 ABI options:	SH64 Options.	(line 25)
* SH64 addressing modes:	SH64-Addressing.	(line 6)
* SH64 ISA options:	SH64 Options.	(line 6)
* SH64 line comment character:	SH64-Chars.	(line 6)
* SH64 line separator:	SH64-Chars.	(line 13)
* SH64 machine directives:	SH64 Directives.	(line 9)
* SH64 opcode summary:	SH64 Opcodes.	(line 6)
* SH64 options:	SH64 Options.	(line 6)
* SH64 registers:	SH64-Regs.	(line 6)
* SH64 support:	SH64-Dependent.	(line 6)
* shigh directive, M32R:	M32R-Directives.	(line 26)
* short directive:	Short.	(line 6)
* short directive, TIC54X:	TIC54X-Directives.	(line 109)
* SIMD, i386:	i386-SIMD.	(line 6)
* SIMD, x86-64:	i386-SIMD.	(line 6)
* single character constant:	Chars.	(line 6)
* single directive:	Single.	(line 6)
* single directive, i386:	i386-Float.	(line 14)
* single directive, x86-64:	i386-Float.	(line 14)
* single quote, Z80:	Z80-Chars.	(line 20)
* sixteen bit integers:	hword.	(line 6)
* sixteen byte integer:	Octa.	(line 6)
* size directive (COFF version):	Size.	(line 11)
* size directive (ELF version):	Size.	(line 19)
* size modifiers, D10V:	D10V-Size.	(line 6)
* size modifiers, D30V:	D30V-Size.	(line 6)
* size modifiers, M680x0:	M68K-Syntax.	(line 8)
* size prefixes, i386:	i386-Prefixes.	(line 27)
* size suffixes, H8/300:	H8/300 Opcodes.	(line 160)
* size, translations, Sparc:	Sparc-Size-Translations.	(line 6)
* sizes operands, i386:	i386-Variations.	(line 28)
* sizes operands, x86-64:	i386-Variations.	(line 28)
* skip directive:	Skip.	(line 6)
* skip directive, M680x0:	M68K-Directives.	(line 19)
* skip directive, SPARC:	Sparc-Directives.	(line 48)
* sleb128 directive:	Sleb128.	(line 6)
* small data, MIPS:	MIPS Small Data.	(line 6)
* SmartMIPS instruction generation override:	MIPS ASE Instruction Generation Overrides.	(line 11)
* SOM symbol attributes:	SOM Symbols.	(line 6)
* source program:	Input Files.	(line 6)
* source, destination operands; i386:	i386-Variations.	(line 21)
* source, destination operands; x86-64:	i386-Variations.	(line 21)
* sp register:	Xtensa Registers.	(line 6)
* sp register, V850:	V850-Regs.	(line 12)
* space directive:	Space.	(line 6)
* space directive, TIC54X:	TIC54X-Directives.	(line 194)
* space used, maximum for assembly:	statistics.	(line 6)
* SPARC architectures:	Sparc-Opts.	(line 6)
* Sparc constants:	Sparc-Constants.	(line 6)
* SPARC data alignment:	Sparc-Aligned-Data.	(line 6)
* SPARC floating point (IEEE):	Sparc-Float.	(line 6)
* Sparc line comment character:	Sparc-Chars.	(line 6)
* Sparc line separator:	Sparc-Chars.	(line 14)
* SPARC machine directives:	Sparc-Directives.	(line 6)

* SPARC options:	Sparc-Opts.	(line 6)
* Sparc registers:	Sparc-Regs.	(line 6)
* Sparc relocations:	Sparc-Relocs.	(line 6)
* Sparc size translations:	Sparc-Size-Translations.	(line 6)
* SPARC support:	Sparc-Dependent.	(line 6)
* SPARC syntax:	Sparc-Aligned-Data.	(line 21)
* special characters, M680x0:	M68K-Chars.	(line 6)
* special purpose registers, MSP 430:	MSP430-Regs.	(line 11)
* ssslist directive, TIC54X:	TIC54X-Directives.	(line 201)
* ssnolist directive, TIC54X:	TIC54X-Directives.	(line 201)
* stabd directive:	Stab.	(line 38)
* stabn directive:	Stab.	(line 49)
* stabs directive:	Stab.	(line 52)
* stabX directives:	Stab.	(line 6)
* stack pointer, ARC:	ARC-Regs.	(line 20)
* standard assembler sections:	Secs Background.	(line 27)
* standard input, as input file:	Command Line.	(line 10)
* statement separator character:	Statements.	(line 6)
* statement separator, AArch64:	AArch64-Chars.	(line 10)
* statement separator, Alpha:	Alpha-Chars.	(line 11)
* statement separator, ARC:	ARC-Chars.	(line 27)
* statement separator, ARM:	ARM-Chars.	(line 14)
* statement separator, AVR:	AVR-Chars.	(line 14)
* statement separator, CR16:	CR16-Chars.	(line 12)
* statement separator, Epiphany:	Epiphany-Chars.	(line 14)
* statement separator, H8/300:	H8/300-Chars.	(line 8)
* statement separator, i386:	i386-Chars.	(line 18)
* statement separator, i860:	i860-Chars.	(line 14)
* statement separator, i960:	i960-Chars.	(line 14)
* statement separator, IA-64:	IA-64-Chars.	(line 8)
* statement separator, IP2K:	IP2K-Chars.	(line 14)
* statement separator, LM32:	LM32-Chars.	(line 12)
* statement separator, M32C:	M32C-Chars.	(line 14)
* statement separator, M68HC11:	M68HC11-Syntax.	(line 26)
* statement separator, Meta:	Meta-Chars.	(line 8)
* statement separator, MicroBlaze:	MicroBlaze-Chars.	(line 14)
* statement separator, MIPS:	MIPS-Chars.	(line 14)
* statement separator, MSP 430:	MSP430-Chars.	(line 14)
* statement separator, NS32K:	NS32K-Chars.	(line 18)
* statement separator, PJ:	PJ-Chars.	(line 14)
* statement separator, PowerPC:	PowerPC-Chars.	(line 18)
* statement separator, RL78:	RL78-Chars.	(line 14)
* statement separator, RX:	RX-Chars.	(line 14)
* statement separator, s390:	s390 Characters.	(line 13)
* statement separator, SCORE:	SCORE-Chars.	(line 14)
* statement separator, SH:	SH-Chars.	(line 8)
* statement separator, SH64:	SH64-Chars.	(line 13)
* statement separator, Sparc:	Sparc-Chars.	(line 14)
* statement separator, TIC54X:	TIC54X-Chars.	(line 17)
* statement separator, TIC6X:	TIC6X Syntax.	(line 13)
* statement separator, V850:	V850-Chars.	(line 13)
* statement separator, VAX:	VAX-Chars.	(line 14)
* statement separator, Visium:	Visium Characters.	(line 14)
* statement separator, XGATE:	XGATE-Syntax.	(line 25)
* statement separator, XStormy16:	XStormy16-Chars.	(line 14)
* statement separator, Z80:	Z80-Chars.	(line 13)
* statement separator, Z8000:	Z8000-Chars.	(line 13)
* statements, structure of:	Statements.	(line 6)


```

* statistics, about assembly:          statistics.          (line 6)
* Status register, ARC:               ARC-Regs.          (line 57)
* STATUS32 saved on exception, ARC:   ARC-Regs.          (line 82)
* stopping the assembly:              Abort.             (line 6)
* Stored STATUS32 register on entry to level P0 interrupts, ARC: ARC-Regs.
                                         (line 69)
* string constants:                   Strings.           (line 6)
* string directive:                   String.            (line 8)
* string directive on HPPA:            HPPA Directives. (line 137)
* string directive, TIC54X:           TIC54X-Directives. (line 206)
* string literals:                    Ascii.             (line 6)
* string, copying to object file:     String.            (line 8)
* string16 directive:                 String.            (line 8)
* string16, copying to object file:   String.            (line 8)
* string32 directive:                 String.            (line 8)
* string32, copying to object file:   String.            (line 8)
* string64 directive:                 String.            (line 8)
* string64, copying to object file:   String.            (line 8)
* string8 directive:                  String.            (line 8)
* string8, copying to object file:    String.            (line 8)
* struct directive:                   Struct.            (line 6)
* struct directive, TIC54X:           TIC54X-Directives. (line 214)
* structure debugging, COFF:          Tag.               (line 6)
* sub-instruction ordering, D10V:     D10V-Chars.       (line 14)
* sub-instruction ordering, D30V:     D30V-Chars.       (line 14)
* sub-instructions, D10V:             D10V-Subs.        (line 6)
* sub-instructions, D30V:             D30V-Subs.        (line 6)
* subexpressions:                     Arguments.         (line 24)
* subsection directive:               SubSection.        (line 6)
* subsym builtins, TIC54X:            TIC54X-Macros.    (line 16)
* subtitles for listings:             Sbtctl.            (line 6)
* subtraction, permitted arguments:   Infix Ops.        (line 50)
* summary of options:                 Overview.          (line 6)
* support:                             HPPA-Dependent.   (line 6)
* supporting files, including:        Include.           (line 6)
* suppressing warnings:               W.                 (line 11)
* sval:                                Z8000 Directives. (line 33)
* symbol attributes:                  Symbol Attributes. (line 6)
* symbol attributes, a.out:           a.out Symbols.    (line 6)
* symbol attributes, COFF:            COFF Symbols.     (line 6)
* symbol attributes, SOM:             SOM Symbols.      (line 6)
* symbol descriptor, COFF:            Desc.              (line 6)
* symbol modifiers:                   AVR-Modifiers.     (line 12)
* symbol modifiers <1>:                LM32-Modifiers.    (line 12)
* symbol modifiers <2>:                M32C-Modifiers.    (line 11)
* symbol modifiers <3>:                M68HC11-Modifiers. (line 12)
* symbol modifiers, TILE-Gx:          TILE-Gx Modifiers. (line 6)
* symbol modifiers, TILEPro:          TILEPro Modifiers. (line 6)
* symbol names:                       Symbol Names.      (line 6)
* symbol names, $ in:                 D10V-Chars.       (line 46)
* symbol names, $ in <1>:              D30V-Chars.       (line 70)
* symbol names, $ in <2>:              Meta-Chars.       (line 10)
* symbol names, $ in <3>:              SH-Chars.         (line 15)
* symbol names, $ in <4>:              SH64-Chars.       (line 15)
* symbol names, local:                 Symbol Names.     (line 30)
* symbol names, temporary:            Symbol Names.     (line 43)
* symbol prefix character, ARC:       ARC-Chars.        (line 20)
* symbol storage class (COFF):        Sc1.               (line 6)
* symbol type:                         Symbol Type.       (line 6)

```

* symbol type, COFF:	Type.	(line 11)
* symbol type, ELF:	Type.	(line 22)
* symbol value:	Symbol Value.	(line 6)
* symbol value, setting:	Set.	(line 6)
* symbol values, assigning:	Setting Symbols.	(line 6)
* symbol versioning:	Symver.	(line 6)
* symbol, common:	Comm.	(line 6)
* symbol, making visible to linker:	Global.	(line 6)
* symbolic debuggers, information for:	Stab.	(line 6)
* symbols:	Symbols.	(line 6)
* Symbols in position-independent code, CRIS:	CRIS-Pic.	(line 6)
* symbols with uppercase, VAX/VMS:	VAX-Opts.	(line 42)
* symbols, assigning values to:	Equ.	(line 6)
* Symbols, built-in, CRIS:	CRIS-Symbols.	(line 6)
* Symbols, CRIS, built-in:	CRIS-Symbols.	(line 6)
* symbols, local common:	Lcomm.	(line 6)
* symver directive:	Symver.	(line 6)
* syntax compatibility, i386:	i386-Variations.	(line 6)
* syntax compatibility, x86-64:	i386-Variations.	(line 6)
* syntax, AVR:	AVR-Modifiers.	(line 6)
* syntax, Blackfin:	Blackfin Syntax.	(line 6)
* syntax, D10V:	D10V-Syntax.	(line 6)
* syntax, D30V:	D30V-Syntax.	(line 6)
* syntax, LM32:	LM32-Modifiers.	(line 6)
* syntax, M680x0:	M68K-Syntax.	(line 8)
* syntax, M68HC11:	M68HC11-Syntax.	(line 6)
* syntax, M68HC11 <1>:	M68HC11-Modifiers.	(line 6)
* syntax, machine-independent:	Syntax.	(line 6)
* syntax, RL78:	RL78-Modifiers.	(line 6)
* syntax, RX:	RX-Modifiers.	(line 6)
* syntax, SPARC:	Sparc-Aligned-Data.	(line 20)
* syntax, TILE-Gx:	TILE-Gx Syntax.	(line 6)
* syntax, TILEPro:	TILEPro Syntax.	(line 6)
* syntax, XGATE:	XGATE-Syntax.	(line 6)
* syntax, Xtensa assembler:	Xtensa Syntax.	(line 6)
* sysproc directive, i960:	Directives-i960.	(line 37)
* tab (\t):	Strings.	(line 27)
* tab directive, TIC54X:	TIC54X-Directives.	(line 245)
* tag directive:	Tag.	(line 6)
* tag directive, TIC54X:	TIC54X-Directives.	(line 214)
* tag directive, TIC54X <1>:	TIC54X-Directives.	(line 248)
* TBM, i386:	i386-TBM.	(line 6)
* TBM, x86-64:	i386-TBM.	(line 6)
* tdaoff pseudo-op, V850:	V850 Opcodes.	(line 81)
* temporary symbol names:	Symbol Names.	(line 43)
* text and data sections, joining:	R.	(line 6)
* text directive:	Text.	(line 6)
* text section:	Ld Sections.	(line 9)
* tfloat directive, i386:	i386-Float.	(line 14)
* tfloat directive, x86-64:	i386-Float.	(line 14)
* Thumb support:	ARM-Dependent.	(line 6)
* TIC54X builtin math functions:	TIC54X-Builtins.	(line 6)
* TIC54X line comment character:	TIC54X-Chars.	(line 6)
* TIC54X line separator:	TIC54X-Chars.	(line 17)
* TIC54X machine directives:	TIC54X-Directives.	(line 6)
* TIC54X memory-mapped registers:	TIC54X-MMRegs.	(line 6)
* TIC54X options:	TIC54X-Opts.	(line 6)
* TIC54X subsym builtins:	TIC54X-Macros.	(line 16)
* TIC54X support:	TIC54X-Dependent.	(line 6)

```

* TIC54X-specific macros:      TIC54X-Macros.      (line 6)
* TIC6X big-endian output:     TIC6X Options.     (line 46)
* TIC6X line comment character: TIC6X Syntax.     (line 6)
* TIC6X line separator:       TIC6X Syntax.     (line 13)
* TIC6X little-endian output:  TIC6X Options.     (line 46)
* TIC6X machine directives:   TIC6X Directives. (line 6)
* TIC6X options:              TIC6X Options.     (line 6)
* TIC6X support:              TIC6X-Dependent.  (line 6)
* TILE-Gx machine directives:  TILE-Gx Directives. (line 6)
* TILE-Gx modifiers:          TILE-Gx Modifiers. (line 6)
* TILE-Gx opcode names:       TILE-Gx Opcodes.  (line 6)
* TILE-Gx register names:     TILE-Gx Registers. (line 6)
* TILE-Gx support:            TILE-Gx-Dependent. (line 6)
* TILE-Gx syntax:             TILE-Gx Syntax.   (line 6)
* TILEPro machine directives:  TILEPro Directives. (line 6)
* TILEPro modifiers:          TILEPro Modifiers. (line 6)
* TILEPro opcode names:       TILEPro Opcodes.  (line 6)
* TILEPro register names:     TILEPro Registers. (line 6)
* TILEPro support:            TILEPro-Dependent. (line 6)
* TILEPro syntax:             TILEPro Syntax.   (line 6)
* time, total for assembly:    statistics.        (line 6)
* title directive:             Title.             (line 6)
* tls_gd directive, Nios II:   Nios II Relocations.
                                (line 38)
* tls_ie directive, Nios II:   Nios II Relocations.
                                (line 38)
* tls_ldm directive, Nios II:  Nios II Relocations.
                                (line 38)
* tls_ldo directive, Nios II:  Nios II Relocations.
                                (line 38)
* tls_le directive, Nios II:   Nios II Relocations.
                                (line 38)
* TMS320C6X support:          TIC6X-Dependent.  (line 6)
* tp register, V850:           V850-Regs.        (line 16)
* transform directive:         Transform Directive.
                                (line 6)
* trusted compiler:           f.                (line 6)
* turning preprocessing on and off: Preprocessing.    (line 26)
* type directive (COFF version): Type.              (line 11)
* type directive (ELF version): Type.              (line 22)
* type of a symbol:           Symbol Type.       (line 6)
* ualong directive, SH:       SH Directives.    (line 6)
* uaqquad directive, SH:      SH Directives.    (line 6)
* uaword directive, SH:       SH Directives.    (line 6)
* ubyte directive, TIC54X:    TIC54X-Directives. (line 34)
* uchar directive, TIC54X:    TIC54X-Directives. (line 34)
* uhalf directive, TIC54X:    TIC54X-Directives. (line 109)
* uint directive, TIC54X:     TIC54X-Directives. (line 109)
* uleb128 directive:          Uleb128.          (line 6)
* ulong directive, TIC54X:    TIC54X-Directives. (line 133)
* undefined section:          Ld Sections.      (line 36)
* union directive, TIC54X:    TIC54X-Directives. (line 248)
* unsegm:                     Z8000 Directives. (line 14)
* usect directive, TIC54X:    TIC54X-Directives. (line 260)
* ushort directive, TIC54X:   TIC54X-Directives. (line 109)
* uword directive, TIC54X:    TIC54X-Directives. (line 109)
* V850 command line options:  V850 Options.     (line 9)
* V850 floating point (IEEE): V850 Floating Point.
                                (line 6)

```

```

* V850 line comment character:      V850-Chars.      (line 6)
* V850 line separator:             V850-Chars.      (line 13)
* V850 machine directives:         V850 Directives. (line 6)
* V850 opcodes:                   V850 Opcodes.    (line 6)
* V850 options (none):             V850 Options.    (line 6)
* V850 register names:            V850-Regs.       (line 6)
* V850 support:                   V850-Dependent.  (line 6)
* val directive:                  Val.              (line 6)
* value attribute, COFF:          Val.              (line 6)
* value of a symbol:              Symbol Value.     (line 6)
* var directive, TIC54X:          TIC54X-Directives. (line 270)
* VAX bitfields not supported:     VAX-no.          (line 6)
* VAX branch improvement:         VAX-branch.      (line 6)
* VAX command-line options ignored: VAX-Opts.        (line 6)
* VAX displacement sizing character: VAX-operands.    (line 12)
* VAX floating point:            VAX-float.       (line 6)
* VAX immediate character:        VAX-operands.    (line 6)
* VAX indirect character:         VAX-operands.    (line 9)
* VAX line comment character:     VAX-Chars.       (line 6)
* VAX line separator:            VAX-Chars.       (line 14)
* VAX machine directives:         VAX-directives.  (line 6)
* VAX opcode mnemonics:          VAX-opcodes.     (line 6)
* VAX operand notation:          VAX-operands.    (line 6)
* VAX register names:            VAX-operands.    (line 17)
* VAX support:                   Vax-Dependent.   (line 6)
* Vax-11 C compatibility:         VAX-Opts.        (line 42)
* VAX/VMS options:              VAX-Opts.        (line 42)
* version directive:             Version.          (line 6)
* version directive, TIC54X:      TIC54X-Directives. (line 274)
* version of assembler:          v.                (line 6)
* versions of symbols:           Symver.           (line 6)
* Virtualization instruction generation override: MIPS ASE Instruction Generation
Overrides.
                                     (line 52)
* visibility:                    Hidden.           (line 6)
* visibility <1>:                 Internal.         (line 6)
* visibility <2>:                 Protected.       (line 6)
* Visium line comment character:   Visium Characters. (line 6)
* Visium line separator:         Visium Characters. (line 14)
* Visium options:                Visium Options.   (line 6)
* Visium registers:              Visium Registers. (line 6)
* Visium support:                Visium-Dependent. (line 6)
* VMS (VAX) options:            VAX-Opts.        (line 42)
* vtable_entry directive:        VTableEntry.     (line 6)
* vtable_inherit directive:      VTableInherit.   (line 6)
* warning directive:             Warning.          (line 6)
* warning for altered difference tables: K.                (line 6)
* warning messages:             Errors.           (line 6)
* warnings, causing error:       W.                (line 16)
* warnings, M32R:               M32R-Warnings.   (line 6)
* warnings, suppressing:         W.                (line 11)
* warnings, switching on:       W.                (line 19)
* weak directive:                Weak.             (line 6)
* weakref directive:            Weakref.          (line 6)
* whitespace:                   Whitespace.       (line 6)
* whitespace, removed by preprocessor: Preprocessing.    (line 7)
* wide floating point directives, VAX: VAX-directives.  (line 9)
* width directive, TIC54X:       TIC54X-Directives. (line 125)
* Width of continuation lines of disassembly output: listing.

```

```

* Width of first line disassembly output: listing. (line 21)
* Width of source line output: listing. (line 16)
* wmsg directive, TIC54X: TIC54X-Directives. (line 28)
* word aligned program counter, ARC: ARC-Regs. (line 75)
* word directive: Word. (line 44)
* word directive, H8/300: H8/300 Directives. (line 6)
* word directive, i386: i386-Float. (line 6)
* word directive, Nios II: Nios II Directives. (line 21)
* word directive, SPARC: Sparc-Directives. (line 13)
* word directive, TIC54X: TIC54X-Directives. (line 51)
* word directive, x86-64: i386-Float. (line 51)
* writing patterns in memory: Fill. (line 109)
* wval: Z8000 Directives. (line 21)
* x86 machine directives: i386-Directives. (line 24)
* x86-64 arch directive: i386-Arch. (line 24)
* x86-64 att_syntax pseudo op: i386-Variations. (line 6)
* x86-64 conversion instructions: i386-Mnemonics. (line 39)
* x86-64 floating point: i386-Float. (line 6)
* x86-64 immediate operands: i386-Variations. (line 15)
* x86-64 instruction naming: i386-Mnemonics. (line 9)
* x86-64 intel_syntax pseudo op: i386-Variations. (line 6)
* x86-64 jump optimization: i386-Jumps. (line 6)
* x86-64 jump, call, return: i386-Variations. (line 40)
* x86-64 jump/call operands: i386-Variations. (line 15)
* x86-64 memory references: i386-Memory. (line 6)
* x86-64 options: i386-Options. (line 6)
* x86-64 register operands: i386-Variations. (line 15)
* x86-64 registers: i386-Regs. (line 6)
* x86-64 sections: i386-Variations. (line 46)
* x86-64 size suffixes: i386-Variations. (line 28)
* x86-64 source, destination operands: i386-Variations. (line 21)
* x86-64 support: i386-Dependent. (line 6)
* x86-64 syntax compatibility: i386-Variations. (line 6)
* xfloat directive, TIC54X: TIC54X-Directives. (line 62)
* XGATE addressing modes: XGATE-Syntax. (line 28)
* XGATE assembler directives: XGATE-Directives. (line 6)
* XGATE floating point: XGATE-Float. (line 6)
* XGATE line comment character: XGATE-Syntax. (line 16)
* XGATE line separator: XGATE-Syntax. (line 25)
* XGATE opcodes: XGATE-opcodes. (line 6)
* XGATE options: XGATE-Opts. (line 6)
* XGATE support: XGATE-Dependent. (line 6)
* XGATE syntax: XGATE-Syntax. (line 6)
* xlong directive, TIC54X: TIC54X-Directives. (line 133)
* XStormy16 comment character: XStormy16-Chars. (line 11)
* XStormy16 line comment character: XStormy16-Chars. (line 6)
* XStormy16 line separator: XStormy16-Chars. (line 14)
* XStormy16 machine directives: XStormy16 Directives. (line 6)
* XStormy16 pseudo-opcodes: XStormy16 Opcodes. (line 6)
* XStormy16 support: XSTORMY16-Dependent. (line 6)
* Xtensa architecture: Xtensa-Dependent. (line 6)
* Xtensa assembler syntax: Xtensa Syntax. (line 6)
* Xtensa directives: Xtensa Directives. (line 6)
* Xtensa opcode names: Xtensa Opcodes. (line 6)
* Xtensa register names: Xtensa Registers. (line 6)
* xword directive, SPARC: Sparc-Directives. (line 55)

```

* Z80 \$:	Z80-Chars.	(line 15)
* Z80 ':	Z80-Chars.	(line 20)
* Z80 floating point:	Z80 Floating Point.	(line 6)
* Z80 line comment character:	Z80-Chars.	(line 6)
* Z80 line separator:	Z80-Chars.	(line 13)
* Z80 options:	Z80 Options.	(line 6)
* Z80 registers:	Z80-Regs.	(line 6)
* Z80 support:	Z80-Dependent.	(line 6)
* Z80 Syntax:	Z80 Options.	(line 40)
* Z80, case sensitivity:	Z80-Case.	(line 6)
* Z80, \:	Z80-Chars.	(line 18)
* Z80-only directives:	Z80 Directives.	(line 9)
* Z800 addressing modes:	Z8000-Addressing.	(line 6)
* Z8000 directives:	Z8000 Directives.	(line 6)
* Z8000 line comment character:	Z8000-Chars.	(line 6)
* Z8000 line separator:	Z8000-Chars.	(line 13)
* Z8000 opcode summary:	Z8000 Opcodes.	(line 6)
* Z8000 options:	Z8000 Options.	(line 6)
* Z8000 registers:	Z8000-Regs.	(line 6)
* Z8000 support:	Z8000-Dependent.	(line 6)
* zdaoff pseudo-op, V850:	V850 Opcodes.	(line 98)
* zero directive:	Zero.	(line 6)
* zero register, V850:	V850-Regs.	(line 7)
* zero-terminated strings:	Asciz.	(line 6)

US
Tag Table:
Node: Top DEL 736
Node: Overview DEL 1721
Node: Manual DEL 40079
Node: GNU Assembler DEL 41023
Node: Object Formats DEL 42194
Node: Command Line DEL 42646
Node: Input Files DEL 43732
Node: Object DEL 45713
Node: Errors DEL 46609
Node: Invoking DEL 48171
Node: a DEL 50177
Node: alternate DEL 52088
Node: D DEL 52260
Node: f DEL 52493
Node: I DEL 53002
Node: K DEL 53546
Node: L DEL 53850
Node: listing DEL 54589
Node: M DEL 56248
Node: MD DEL 60647
Node: no-pad-sections DEL 61087
Node: o DEL 61462
Node: R DEL 61930
Node: statistics DEL 62960
Node: traditional-format DEL 63367
Node: v DEL 63840
Node: W DEL 64115
Node: Z DEL 65022
Node: Syntax DEL 65544
Node: Preprocessing DEL 66136
Node: Whitespace DEL 67700

Node: Comments^{DEL}68096
Node: Symbol Intro^{DEL}70107
Node: Statements^{DEL}71092
Node: Constants^{DEL}73080
Node: Characters^{DEL}73711
Node: Strings^{DEL}74213
Node: Chars^{DEL}76387
Node: Numbers^{DEL}77239
Node: Integers^{DEL}77779
Node: Bignums^{DEL}78435
Node: Flonums^{DEL}78791
Node: Sections^{DEL}80535
Node: Secs Background^{DEL}80913
Node: Ld Sections^{DEL}85946
Node: As Sections^{DEL}88330
Node: Sub-Sections^{DEL}89240
Node: bss^{DEL}92388
Node: Symbols^{DEL}93338
Node: Labels^{DEL}93986
Node: Setting Symbols^{DEL}94717
Node: Symbol Names^{DEL}95271
Node: Dot^{DEL}100738
Node: Symbol Attributes^{DEL}101185
Node: Symbol Value^{DEL}101918
Node: Symbol Type^{DEL}102963
Node: a.out Symbols^{DEL}103351
Node: Symbol Desc^{DEL}103613
Node: Symbol Other^{DEL}103909
Node: COFF Symbols^{DEL}104078
Node: SOM Symbols^{DEL}104751
Node: Expressions^{DEL}105193
Node: Empty Exprs^{DEL}105942
Node: Integer Exprs^{DEL}106289
Node: Arguments^{DEL}106684
Node: Operators^{DEL}107790
Node: Prefix Ops^{DEL}108125
Node: Infix Ops^{DEL}108452
Node: Pseudo Ops^{DEL}110846
Node: Abort^{DEL}116492
Node: ABORT (COFF)^{DEL}116905
Node: Align^{DEL}117113
Node: Altmacro^{DEL}119395
Node: Ascii^{DEL}120726
Node: Asciz^{DEL}121035
Node: Balign^{DEL}121280
Node: Bundle directives^{DEL}123156
Node: Byte^{DEL}126137
Node: CFI directives^{DEL}126396
Node: Comm^{DEL}135709
Ref: Comm-Footnote-1^{DEL}137309
Node: Data^{DEL}137671
Node: Def^{DEL}137988
Node: Desc^{DEL}138220
Node: Dim^{DEL}138720
Node: Double^{DEL}138977
Node: Eject^{DEL}139315
Node: Else^{DEL}139490
Node: Elseif^{DEL}139790
Node: End^{DEL}140084

Node: Endef^{DEL}140299
Node: Endfunc^{DEL}140476
Node: Endif^{DEL}140651
Node: Equ^{DEL}140912
Node: Equiv^{DEL}141426
Node: Eqv^{DEL}141982
Node: Err^{DEL}142346
Node: Error^{DEL}142657
Node: Exitm^{DEL}143102
Node: Extern^{DEL}143271
Node: Fail^{DEL}143532
Node: File^{DEL}143977
Node: Fill^{DEL}145306
Node: Float^{DEL}146270
Node: Func^{DEL}146612
Node: Global^{DEL}147202
Node: Gnu_attribute^{DEL}147959
Node: Hidden^{DEL}148184
Node: hword^{DEL}148770
Node: Ident^{DEL}149098
Node: If^{DEL}149672
Node: Incbin^{DEL}152733
Node: Include^{DEL}153427
Node: Int^{DEL}153978
Node: Internal^{DEL}154359
Node: Irp^{DEL}155006
Node: Irpc^{DEL}155885
Node: Lcomm^{DEL}156801
Node: Lflags^{DEL}157548
Node: Line^{DEL}157742
Node: Linkonce^{DEL}158658
Node: List^{DEL}159887
Node: Ln^{DEL}160494
Node: Loc^{DEL}160644
Node: Loc_mark_labels^{DEL}162029
Node: Local^{DEL}162513
Node: Long^{DEL}163125
Node: Macro^{DEL}163303
Node: MRI^{DEL}169234
Node: Noaltmacro^{DEL}169572
Node: Nolist^{DEL}169741
Node: Octa^{DEL}170170
Node: Offset^{DEL}170507
Node: Org^{DEL}170834
Node: P2align^{DEL}172119
Node: PopSection^{DEL}174047
Node: Previous^{DEL}174555
Node: Print^{DEL}175967
Node: Protected^{DEL}176196
Node: Psize^{DEL}176843
Node: Purgem^{DEL}177527
Node: PushSection^{DEL}177748
Node: Quad^{DEL}178492
Node: Reloc^{DEL}178946
Node: Rept^{DEL}179707
Node: Sbt^{DEL}180121
Node: Sc^{DEL}180486
Node: Section^{DEL}180827
Ref: Section Name Substitutions^{DEL}182885

Node: Set~~DEL~~188954
Node: Short~~DEL~~190036
Node: Single~~DEL~~190357
Node: Size~~DEL~~190702
Node: Skip~~DEL~~191368
Node: Sleb128~~DEL~~191690
Node: Space~~DEL~~192012
Node: Stab~~DEL~~192651
Node: String~~DEL~~194654
Node: Struct~~DEL~~195646
Node: SubSection~~DEL~~196369
Node: Symver~~DEL~~196930
Node: Tag~~DEL~~199324
Node: Text~~DEL~~199704
Node: Title~~DEL~~200024
Node: Type~~DEL~~200403
Node: Uleb128~~DEL~~202707
Node: Val~~DEL~~203029
Node: Version~~DEL~~203277
Node: VTableEntry~~DEL~~203549
Node: VTableInherit~~DEL~~203837
Node: Warning~~DEL~~204285
Node: Weak~~DEL~~204519
Node: Weakref~~DEL~~205188
Node: Word~~DEL~~206153
Node: Zero~~DEL~~207993
Node: Deprecated~~DEL~~208405
Node: Object Attributes~~DEL~~208639
Node: GNU Object Attributes~~DEL~~210359
Node: Defining New Object Attributes~~DEL~~213884
Node: Machine Dependencies~~DEL~~214676
Node: AArch64-Dependent~~DEL~~218633
Node: AArch64 Options~~DEL~~219115
Node: AArch64 Extensions~~DEL~~221749
Node: AArch64 Syntax~~DEL~~224460
Node: AArch64-Chars~~DEL~~224760
Node: AArch64-Regs~~DEL~~225246
Node: AArch64-Relocations~~DEL~~225540
Node: AArch64 Floating Point~~DEL~~226614
Node: AArch64 Directives~~DEL~~226839
Node: AArch64 Opcodes~~DEL~~229725
Node: AArch64 Mapping Symbols~~DEL~~230402
Node: Alpha-Dependent~~DEL~~230783
Node: Alpha Notes~~DEL~~231223
Node: Alpha Options~~DEL~~231504
Node: Alpha Syntax~~DEL~~233979
Node: Alpha-Chars~~DEL~~234448
Node: Alpha-Regs~~DEL~~234860
Node: Alpha-Relocs~~DEL~~235247
Node: Alpha Floating Point~~DEL~~241493
Node: Alpha Directives~~DEL~~241715
Node: Alpha Opcodes~~DEL~~247242
Node: ARC-Dependent~~DEL~~247537
Node: ARC Options~~DEL~~247982
Node: ARC Syntax~~DEL~~251108
Node: ARC-Chars~~DEL~~251336
Node: ARC-Regs~~DEL~~252458
Node: ARC Directives~~DEL~~255226
Node: ARC Modifiers~~DEL~~263276

Node: ARC Symbols~~DEL~~264286
Node: ARC Opcodes~~DEL~~264840
Node: ARM-Dependent~~DEL~~265086
Node: ARM Options~~DEL~~265551
Node: ARM Syntax~~DEL~~275645
Node: ARM-Instruction-Set~~DEL~~276013
Node: ARM-Chars~~DEL~~277228
Node: ARM-Regs~~DEL~~277939
Node: ARM-Relocations~~DEL~~278148
Node: ARM-Neon-Alignment~~DEL~~279834
Node: ARM Floating Point~~DEL~~280299
Node: ARM Directives~~DEL~~280498
Ref: arm_fnend~~DEL~~284938
Ref: arm_fnstart~~DEL~~285257
Ref: arm_pad~~DEL~~287664
Ref: arm_save~~DEL~~288266
Ref: arm_setfp~~DEL~~288967
Node: ARM Opcodes~~DEL~~292253
Node: ARM Mapping Symbols~~DEL~~294340
Node: ARM Unwinding Tutorial~~DEL~~295149
Node: AVR-Dependent~~DEL~~301349
Node: AVR Options~~DEL~~301639
Node: AVR Syntax~~DEL~~307676
Node: AVR-Chars~~DEL~~307963
Node: AVR-Regs~~DEL~~308522
Node: AVR-Modifiers~~DEL~~309103
Node: AVR Opcodes~~DEL~~311167
Node: Blackfin-Dependent~~DEL~~316413
Node: Blackfin Options~~DEL~~316725
Node: Blackfin Syntax~~DEL~~317699
Node: Blackfin Directives~~DEL~~323908
Node: CR16-Dependent~~DEL~~324649
Node: CR16 Operand Qualifiers~~DEL~~324949
Node: CR16 Syntax~~DEL~~327658
Node: CR16-Chars~~DEL~~327844
Node: CRIS-Dependent~~DEL~~328381
Node: CRIS-Opts~~DEL~~328727
Ref: march-option~~DEL~~330412
Node: CRIS-Expand~~DEL~~332231
Node: CRIS-Symbols~~DEL~~333414
Node: CRIS-Syntax~~DEL~~334585
Node: CRIS-Chars~~DEL~~334921
Node: CRIS-Pic~~DEL~~335472
Ref: crispic~~DEL~~335667
Node: CRIS-Regs~~DEL~~339216
Node: CRIS-Pseudos~~DEL~~339633
Ref: crisnous~~DEL~~340410
Node: D10V-Dependent~~DEL~~341698
Node: D10V-Opts~~DEL~~342049
Node: D10V-Syntax~~DEL~~343018
Node: D10V-Size~~DEL~~343547
Node: D10V-Subs~~DEL~~344518
Node: D10V-Chars~~DEL~~345553
Node: D10V-Regs~~DEL~~347364
Node: D10V-Addressing~~DEL~~348391
Node: D10V-Word~~DEL~~349069
Node: D10V-Float~~DEL~~349584
Node: D10V-Opcodes~~DEL~~349895
Node: D30V-Dependent~~DEL~~350288

Node: D30V-Opts^{DEL}350645
Node: D30V-Syntax^{DEL}351323
Node: D30V-Size^{DEL}351857
Node: D30V-Subs^{DEL}352830
Node: D30V-Chars^{DEL}353867
Node: D30V-Guarded^{DEL}356477
Node: D30V-Regs^{DEL}357154
Node: D30V-Addressing^{DEL}358273
Node: D30V-Float^{DEL}358935
Node: D30V-Opcodes^{DEL}359248
Node: Epiphany-Dependent^{DEL}359643
Node: Epiphany Options^{DEL}359931
Node: Epiphany Syntax^{DEL}360330
Node: Epiphany-Chars^{DEL}360531
Node: H8/300-Dependent^{DEL}361085
Node: H8/300 Options^{DEL}361501
Node: H8/300 Syntax^{DEL}361941
Node: H8/300-Chars^{DEL}362242
Node: H8/300-Regs^{DEL}362541
Node: H8/300-Addressing^{DEL}363460
Node: H8/300 Floating Point^{DEL}364484
Node: H8/300 Directives^{DEL}364811
Node: H8/300 Opcodes^{DEL}365939
Node: HPPA-Dependent^{DEL}374258
Node: HPPA Notes^{DEL}374693
Node: HPPA Options^{DEL}375451
Node: HPPA Syntax^{DEL}375646
Node: HPPA Floating Point^{DEL}376916
Node: HPPA Directives^{DEL}377122
Node: HPPA Opcodes^{DEL}385804
Node: ESA/390-Dependent^{DEL}386063
Node: ESA/390 Notes^{DEL}386523
Node: ESA/390 Options^{DEL}387314
Node: ESA/390 Syntax^{DEL}387524
Node: ESA/390 Floating Point^{DEL}389697
Node: ESA/390 Directives^{DEL}389976
Node: ESA/390 Opcodes^{DEL}393266
Node: i386-Dependent^{DEL}393528
Node: i386-Options^{DEL}394855
Node: i386-Directives^{DEL}403463
Node: i386-Syntax^{DEL}404200
Node: i386-Variations^{DEL}404505
Node: i386-Chars^{DEL}407042
Node: i386-Mnemonics^{DEL}407771
Node: i386-Regs^{DEL}411134
Node: i386-Prefixes^{DEL}413964
Node: i386-Memory^{DEL}416724
Node: i386-Jumps^{DEL}419663
Node: i386-Float^{DEL}420786
Node: i386-SIMD^{DEL}422616
Node: i386-LWP^{DEL}423725
Node: i386-BMI^{DEL}424559
Node: i386-TBM^{DEL}424937
Node: i386-16bit^{DEL}425467
Node: i386-Arch^{DEL}427538
Node: i386-Bugs^{DEL}430716
Node: i386-Notes^{DEL}431467
Node: i860-Dependent^{DEL}432326
Node: Notes-i860^{DEL}432766

Node: Options-i860^{DEL}433673
Node: Directives-i860^{DEL}435033
Node: Opcodes for i860^{DEL}436105
Node: Syntax of i860^{DEL}438294
Node: i860-Chars^{DEL}438478
Node: i960-Dependent^{DEL}439037
Node: Options-i960^{DEL}439484
Node: Floating Point-i960^{DEL}443370
Node: Directives-i960^{DEL}443638
Node: Opcodes for i960^{DEL}445672
Node: callj-i960^{DEL}446312
Node: Compare-and-branch-i960^{DEL}446802
Node: Syntax of i960^{DEL}448706
Node: i960-Chars^{DEL}448906
Node: IA-64-Dependent^{DEL}449449
Node: IA-64 Options^{DEL}449750
Node: IA-64 Syntax^{DEL}452901
Node: IA-64-Chars^{DEL}453307
Node: IA-64-Regs^{DEL}453537
Node: IA-64-Bits^{DEL}454463
Node: IA-64-Relocs^{DEL}454993
Node: IA-64 Opcodes^{DEL}455464
Node: IP2K-Dependent^{DEL}455736
Node: IP2K-Opts^{DEL}456008
Node: IP2K-Syntax^{DEL}456507
Node: IP2K-Chars^{DEL}456681
Node: LM32-Dependent^{DEL}457224
Node: LM32 Options^{DEL}457519
Node: LM32 Syntax^{DEL}458152
Node: LM32-Regs^{DEL}458448
Node: LM32-Modifiers^{DEL}459389
Node: LM32-Chars^{DEL}460769
Node: LM32 Opcodes^{DEL}461277
Node: M32C-Dependent^{DEL}461581
Node: M32C-Opts^{DEL}462087
Node: M32C-Syntax^{DEL}462506
Node: M32C-Modifiers^{DEL}462741
Node: M32C-Chars^{DEL}464533
Node: M32R-Dependent^{DEL}465099
Node: M32R-Opts^{DEL}465420
Node: M32R-Directives^{DEL}469582
Node: M32R-Warnings^{DEL}473556
Node: M68K-Dependent^{DEL}476561
Node: M68K-Opts^{DEL}477028
Node: M68K-Syntax^{DEL}484450
Node: M68K-Moto-Syntax^{DEL}486290
Node: M68K-Float^{DEL}488874
Node: M68K-Directives^{DEL}489394
Node: M68K-opcodes^{DEL}490721
Node: M68K-Branch^{DEL}490947
Node: M68K-Chars^{DEL}495142
Node: M68HC11-Dependent^{DEL}496005
Node: M68HC11-Opts^{DEL}496536
Node: M68HC11-Syntax^{DEL}500848
Node: M68HC11-Modifiers^{DEL}503642
Node: M68HC11-Directives^{DEL}505469
Node: M68HC11-Float^{DEL}506843
Node: M68HC11-opcodes^{DEL}507371
Node: M68HC11-Branch^{DEL}507553

Node: Meta-Dependent~~DEL~~510003
Node: Meta Options~~DEL~~510288
Node: Meta Syntax~~DEL~~510950
Node: Meta-Chars~~DEL~~511162
Node: Meta-Regs~~DEL~~511462
Node: MicroBlaze-Dependent~~DEL~~511738
Node: MicroBlaze Directives~~DEL~~512425
Node: MicroBlaze Syntax~~DEL~~513816
Node: MicroBlaze-Chars~~DEL~~514048
Node: MIPS-Dependent~~DEL~~514600
Node: MIPS Options~~DEL~~516034
Node: MIPS Macros~~DEL~~532723
Ref: MIPS Macros-Footer-1~~DEL~~535437
Node: MIPS Symbol Sizes~~DEL~~535580
Node: MIPS Small Data~~DEL~~537252
Node: MIPS ISA~~DEL~~539416
Node: MIPS assembly options~~DEL~~541200
Node: MIPS autoextend~~DEL~~542333
Node: MIPS insn~~DEL~~543067
Node: MIPS FP ABIs~~DEL~~544348
Node: MIPS FP ABI History~~DEL~~544800
Node: MIPS FP ABI Variants~~DEL~~545560
Node: MIPS FP ABI Selection~~DEL~~548113
Node: MIPS FP ABI Compatibility~~DEL~~549176
Node: MIPS NaN Encodings~~DEL~~549986
Node: MIPS Option Stack~~DEL~~551949
Node: MIPS ASE Instruction Generation Overrides~~DEL~~552734
Node: MIPS Floating-Point~~DEL~~555747
Node: MIPS Syntax~~DEL~~556653
Node: MIPS-Chars~~DEL~~556915
Node: MMIX-Dependent~~DEL~~557457
Node: MMIX-Opts~~DEL~~557837
Node: MMIX-Expand~~DEL~~561442
Node: MMIX-Syntax~~DEL~~562754
Ref: mmixsite~~DEL~~563110
Node: MMIX-Chars~~DEL~~563952
Node: MMIX-Symbols~~DEL~~564825
Node: MMIX-Regs~~DEL~~566896
Node: MMIX-Pseudos~~DEL~~567921
Ref: MMIX-loc~~DEL~~568063
Ref: MMIX-local~~DEL~~569144
Ref: MMIX-is~~DEL~~569677
Ref: MMIX-greg~~DEL~~569949
Ref: GREG-base~~DEL~~570867
Ref: MMIX-byte~~DEL~~572186
Ref: MMIX-constants~~DEL~~572658
Ref: MMIX-prefix~~DEL~~573300
Ref: MMIX-spec~~DEL~~573675
Node: MMIX-mmixal~~DEL~~574009
Node: MSP430-Dependent~~DEL~~577504
Node: MSP430 Options~~DEL~~577973
Node: MSP430 Syntax~~DEL~~581153
Node: MSP430-Macros~~DEL~~581469
Node: MSP430-Chars~~DEL~~582199
Node: MSP430-Regs~~DEL~~582914
Node: MSP430-Ext~~DEL~~583475
Node: MSP430 Floating Point~~DEL~~585294
Node: MSP430 Directives~~DEL~~585518
Node: MSP430 Opcodes~~DEL~~586833

Node: MSP430 Profiling Capability^{DEL}587228
Node: NDS32-Dependent^{DEL}589556
Node: NDS32 Options^{DEL}590165
Node: NDS32 Syntax^{DEL}592048
Node: NDS32-Chars^{DEL}592316
Node: NDS32-Regs^{DEL}592783
Node: NDS32-Ops^{DEL}593637
Node: NiosII-Dependent^{DEL}597234
Node: Nios II Options^{DEL}597653
Node: Nios II Syntax^{DEL}598885
Node: Nios II Chars^{DEL}599091
Node: Nios II Relocations^{DEL}599282
Node: Nios II Directives^{DEL}600853
Node: Nios II Opcodes^{DEL}602415
Node: NS32K-Dependent^{DEL}602690
Node: NS32K Syntax^{DEL}602917
Node: NS32K-Chars^{DEL}603066
Node: PDP-11-Dependent^{DEL}603806
Node: PDP-11-Options^{DEL}604196
Node: PDP-11-Pseudos^{DEL}609255
Node: PDP-11-Syntax^{DEL}609600
Node: PDP-11-Mnemonics^{DEL}610432
Node: PDP-11-Synthetic^{DEL}610734
Node: PJ-Dependent^{DEL}610952
Node: PJ Options^{DEL}611215
Node: PJ Syntax^{DEL}611510
Node: PJ-Chars^{DEL}611675
Node: PPC-Dependent^{DEL}612224
Node: PowerPC-Opts^{DEL}612557
Node: PowerPC-Pseudo^{DEL}616183
Node: PowerPC-Syntax^{DEL}616804
Node: PowerPC-Chars^{DEL}616994
Node: RL78-Dependent^{DEL}617745
Node: RL78-Opts^{DEL}618147
Node: RL78-Modifiers^{DEL}618981
Node: RL78-Directives^{DEL}619760
Node: RL78-Syntax^{DEL}620364
Node: RL78-Chars^{DEL}620560
Node: RISC-V-Dependent^{DEL}621116
Node: RISC-V-Opts^{DEL}621338
Node: RX-Dependent^{DEL}621987
Node: RX-Opts^{DEL}622420
Node: RX-Modifiers^{DEL}626663
Node: RX-Directives^{DEL}627766
Node: RX-Float^{DEL}628505
Node: RX-Syntax^{DEL}629140
Node: RX-Chars^{DEL}629319
Node: S/390-Dependent^{DEL}629871
Node: s390 Options^{DEL}630690
Node: s390 Characters^{DEL}632807
Node: s390 Syntax^{DEL}633328
Node: s390 Register^{DEL}634230
Node: s390 Mnemonics^{DEL}635046
Node: s390 Operands^{DEL}638068
Node: s390 Formats^{DEL}640698
Node: s390 Aliases^{DEL}648545
Node: s390 Operand Modifier^{DEL}652510
Node: s390 Instruction Marker^{DEL}656313
Node: s390 Literal Pool Entries^{DEL}657327

Node: s390 Directives~~DEL~~659260
Node: s390 Floating Point~~DEL~~664712
Node: SCORE-Dependent~~DEL~~665160
Node: SCORE-Opts~~DEL~~665462
Node: SCORE-Pseudo~~DEL~~666749
Node: SCORE-Syntax~~DEL~~668831
Node: SCORE-Chars~~DEL~~669013
Node: SH-Dependent~~DEL~~669571
Node: SH Options~~DEL~~669982
Node: SH Syntax~~DEL~~671033
Node: SH-Chars~~DEL~~671306
Node: SH-Regs~~DEL~~671849
Node: SH-Addressing~~DEL~~672463
Node: SH Floating Point~~DEL~~673371
Node: SH Directives~~DEL~~674468
Node: SH Opcodes~~DEL~~674869
Node: SH64-Dependent~~DEL~~679190
Node: SH64 Options~~DEL~~679552
Node: SH64 Syntax~~DEL~~681343
Node: SH64-Chars~~DEL~~681626
Node: SH64-Regs~~DEL~~682175
Node: SH64-Addressing~~DEL~~683271
Node: SH64 Directives~~DEL~~684456
Node: SH64 Opcodes~~DEL~~685440
Node: Sparc-Dependent~~DEL~~686155
Node: Sparc-Opts~~DEL~~686566
Node: Sparc-Aligned-Data~~DEL~~691906
Node: Sparc-Syntax~~DEL~~692738
Node: Sparc-Chars~~DEL~~693312
Node: Sparc-Regs~~DEL~~693875
Node: Sparc-Constants~~DEL~~699723
Node: Sparc-Relocs~~DEL~~704483
Node: Sparc-Size-Translations~~DEL~~709601
Node: Sparc-Float~~DEL~~711251
Node: Sparc-Directives~~DEL~~711446
Node: TIC54X-Dependent~~DEL~~713408
Node: TIC54X-Opts~~DEL~~714171
Node: TIC54X-Block~~DEL~~715212
Node: TIC54X-Env~~DEL~~715572
Node: TIC54X-Constants~~DEL~~715920
Node: TIC54X-Subsyms~~DEL~~716317
Node: TIC54X-Locals~~DEL~~718221
Node: TIC54X-Builtins~~DEL~~718961
Node: TIC54X-Ext~~DEL~~721375
Node: TIC54X-Directives~~DEL~~721946
Node: TIC54X-Macros~~DEL~~732854
Node: TIC54X-MMRegs~~DEL~~734941
Node: TIC54X-Syntax~~DEL~~735178
Node: TIC54X-Chars~~DEL~~735368
Node: TIC6X-Dependent~~DEL~~736059
Node: TIC6X Options~~DEL~~736362
Node: TIC6X Syntax~~DEL~~738361
Node: TIC6X Directives~~DEL~~739464
Node: TILE-Gx-Dependent~~DEL~~741749
Node: TILE-Gx Options~~DEL~~742059
Node: TILE-Gx Syntax~~DEL~~742408
Node: TILE-Gx Opcodes~~DEL~~744644
Node: TILE-Gx Registers~~DEL~~744932
Node: TILE-Gx Modifiers~~DEL~~745703

Node: TILE-Gx Directives^{DEL}750701
Node: TILEPro-Dependent^{DEL}751604
Node: TILEPro Options^{DEL}751913
Node: TILEPro Syntax^{DEL}752097
Node: TILEPro Opcodes^{DEL}754333
Node: TILEPro Registers^{DEL}754624
Node: TILEPro Modifiers^{DEL}755394
Node: TILEPro Directives^{DEL}760182
Node: V850-Dependent^{DEL}761085
Node: V850 Options^{DEL}761481
Node: V850 Syntax^{DEL}765759
Node: V850-Chars^{DEL}765999
Node: V850-Regs^{DEL}766543
Node: V850 Floating Point^{DEL}768053
Node: V850 Directives^{DEL}768259
Node: V850 Opcodes^{DEL}770325
Node: Vax-Dependent^{DEL}776204
Node: VAX-Opts^{DEL}776788
Node: VAX-float^{DEL}780509
Node: VAX-directives^{DEL}781142
Node: VAX-opcodes^{DEL}782002
Node: VAX-branch^{DEL}782391
Node: VAX-operands^{DEL}784895
Node: VAX-no^{DEL}785658
Node: VAX-Syntax^{DEL}785914
Node: VAX-Chars^{DEL}786080
Node: Visium-Dependent^{DEL}786634
Node: Visium Options^{DEL}786941
Node: Visium Syntax^{DEL}787407
Node: Visium Characters^{DEL}787652
Node: Visium Registers^{DEL}788233
Node: Visium Opcodes^{DEL}788505
Node: XGATE-Dependent^{DEL}788931
Node: XGATE-Opts^{DEL}789353
Node: XGATE-Syntax^{DEL}790342
Node: XGATE-Directives^{DEL}792423
Node: XGATE-Float^{DEL}792662
Node: XGATE-opcodes^{DEL}793159
Node: XSTORMY16-Dependent^{DEL}793271
Node: XStormy16 Syntax^{DEL}793617
Node: XStormy16-Chars^{DEL}793807
Node: XStormy16 Directives^{DEL}794420
Node: XStormy16 Opcodes^{DEL}795074
Node: Xtensa-Dependent^{DEL}796129
Node: Xtensa Options^{DEL}796860
Node: Xtensa Syntax^{DEL}801127
Node: Xtensa Opcodes^{DEL}803271
Node: Xtensa Registers^{DEL}805064
Node: Xtensa Optimizations^{DEL}805697
Node: Density Instructions^{DEL}806149
Node: Xtensa Automatic Alignment^{DEL}807251
Node: Xtensa Relaxation^{DEL}809698
Node: Xtensa Branch Relaxation^{DEL}810663
Node: Xtensa Call Relaxation^{DEL}812035
Node: Xtensa Jump Relaxation^{DEL}813816
Node: Xtensa Immediate Relaxation^{DEL}815916
Node: Xtensa Directives^{DEL}818491
Node: Schedule Directive^{DEL}820199
Node: Longcalls Directive^{DEL}820539

Node: Transform Directive^{DEL}821083
Node: Literal Directive^{DEL}821825
Ref: Literal Directive-Footer-1^{DEL}825364
Node: Literal Position Directive^{DEL}825506
Node: Literal Prefix Directive^{DEL}827205
Node: Absolute Literals Directive^{DEL}828103
Node: Z80-Dependent^{DEL}829410
Node: Z80 Options^{DEL}829798
Node: Z80 Syntax^{DEL}831217
Node: Z80-Chars^{DEL}831889
Node: Z80-Regs^{DEL}832740
Node: Z80-Case^{DEL}833092
Node: Z80 Floating Point^{DEL}833537
Node: Z80 Directives^{DEL}833731
Node: Z80 Opcodes^{DEL}835356
Node: Z8000-Dependent^{DEL}836702
Node: Z8000 Options^{DEL}837638
Node: Z8000 Syntax^{DEL}837855
Node: Z8000-Chars^{DEL}838145
Node: Z8000-Regs^{DEL}838627
Node: Z8000-Addressing^{DEL}839417
Node: Z8000 Directives^{DEL}840527
Node: Z8000 Opcodes^{DEL}842136
Node: Reporting Bugs^{DEL}852078
Node: Bug Criteria^{DEL}852804
Node: Bug Reporting^{DEL}853571
Node: Acknowledgements^{DEL}860215
Ref: Acknowledgements-Footer-1^{DEL}865181
Node: GNU Free Documentation License^{DEL}865207
Node: AS Index^{DEL}890357

^{US}

End Tag Table