@c Copyright 1991, 1992, 1993, 1994, 1995, 1997, 1999, 2000, 2001, @c 2002, 2003, 2004 @c Free Software Foundation, Inc. @c This is part of the GAS manual. @c For copying conditions, see the file as.texinfo. @ifset GENERIC @page @node MIPS-Dependent @chapter MIPS Dependent Features @end ifset @ifclear GENERIC @node Machine Dependencies @chapter MIPS Dependent Features @end ifclear

@cindex MIPS processor @scgnu @code@valueAS for @scmips architectures supports several different @scmips processors, and MIPS ISA levels I through V, MIPS32, and MIPS64. For information about the @scmips instruction set, see @citeMIPS RISC Architecture, by Kane and Heindrich (Prentice-Hall). For an overview of @scmips assembly conventions, see "Appendix D: Assembly Language Programming" in the same work.

@menu * MIPS Opts:: Assembler options * MIPS Object:: ECOFF object code * MIPS Stabs:: Directives for debugging information * MIPS ISA:: Directives to override the ISA level * MIPS symbol sizes:: Directives to override the size of symbols * MIPS autoextend:: Directives for extending MIPS 16 bit instructions * MIPS insn:: Directive to mark data as an instruction * MIPS option stack:: Directives to save and restore options * MIPS ASE instruction generation overrides:: Directives to control generation of MIPS ASE instructions @end menu

@node MIPS Opts @section Assembler options

The @scmips configurations of @scgnu @code@valueAS support these special options:

@table @code @cindex @code-G option (MIPS) @item -G @varnum This option sets the largest size of an object that can be referenced implicitly with the @codegp register. It is only accepted for targets that use @scecoff format. The default value is 8.

@cindex @code-EB option (MIPS) @cindex @code-EL option (MIPS) @cindex MIPS big-endian output @cindex MIPS little-endian output @cindex big-endian output, MIPS @cindex little-endian output, MIPS @item -EB @itemx -EL Any @scmips configuration of @code@valueAS can select big-endian or little-endian output at run time (unlike the other @scgnu development tools, which must be configured for one or the other). Use @samp-EB to select big-endian output, and @samp-EL for little-endian.

@item -KPIC @cindex PIC selection, MIPS @cindex @option-KPIC option, MIPS 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.

@item -mvxworks-pic @cindex @option-mvxworks-pic option, MIPS Generate VxWorks PIC. This option tells the assembler to generate VxWorks-style position-independent macro expansions.

@cindex MIPS architecture options @item -mips1 @itemx -mips2 @itemx -mips3 @itemx -mips4 @itemx -mips5 @itemx -mips32 @itemx -mips32r2 @itemx -mips64 @itemx -mips64r2 Generate code for a particular MIPS Instruction Set Architecture level. @samp-mips1 corresponds to the @scr2000 and @scr3000 processors, @samp-mips2 to the @scr6000 processor, @samp-mips3 to the @scr4000 processor, and @samp-mips4 to the @scr8000 and @scr10000 processors. @samp-mips5, @samp-mips32, @samp-mips32r2, @samp-mips64, and @samp-mips64r2 correspond to generic @scMIPS V, @scMIPS32, @scMIPS32 Release 2, @scMIPS64, and @scMIPS64 Release 2 ISA processors, respectively. You can also switch instruction sets during the assembly; see @refMIPS ISA, Directives to override the ISA level.

@item -mgp32 @itemx -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. @samp-mgp32 controls the size of general-purpose registers and @samp-mfp32 controls the size of floating-point registers.

The @code.set gp=32 and @code.set fp=32 directives allow the size of registers to be changed for parts of an object. The default value is restored by @code.set gp=default and @code.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.

@item -mgp64 @itemx -mfp64 Assume that 64-bit registers are available. This is provided in the interests of symmetry with @samp-mgp32 and @samp-mfp32.

The @code.set gp=64 and @code.set fp=64 directives allow the size of registers to be changed for parts of an object. The default value is restored by @code.set gp=default and @code.set fp=default.

@item -mips16 @itemx -no-mips16 Generate code for the MIPS 16 processor. This is equivalent to putting @code.set mips16 at the start of the assembly file. @samp-no-mips16 turns off this option.

@item -msmartmips @itemx -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 @code.set smartmips at the start of the assembly file. @samp-mno-smartmips turns off this option.

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

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

@item -mdsp @itemx -mno-dsp Generate code for the DSP Release 1 Application Specific Extension. This tells the assembler to accept DSP Release 1 instructions. @samp-mno-dsp turns off this option.

@item -mdspr2 @itemx -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. @samp-mno-dspr2 turns off this option.

@item -mmt @itemx -mno-mt Generate code for the MT Application Specific Extension. This tells the assembler to accept MT instructions. @samp-mno-mt turns off this option.

@item -mfix7000 @itemx -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.

@item -mfix-vr4120 @itemx -no-mfix-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.

@item -mfix-vr4130 @itemx -no-mfix-vr4130 Insert nops to work around the VR4130 @sampmflo/@sampmfhi errata.

@item -m4010 @itemx -no-m4010 Generate code for the LSI @scr4010 chip. This tells the assembler to accept the @scr4010 specific instructions (@sampaddciu, @sampffc, etc.), and to not schedule @sampnop instructions around accesses to the @sampHI and @sampLO registers. @samp-no-m4010 turns off this option.

@item -m4650 @itemx -no-m4650 Generate code for the MIPS @scr4650 chip. This tells the assembler to accept the @sampmad and @sampmadu instruction, and to not schedule @sampnop instructions around accesses to the @sampHI and @sampLO registers. @samp-no-m4650 turns off this option.

@itemx -m3900 @itemx -no-m3900 @itemx -m4100 @itemx -no-m4100 For each option @samp-m@varnnnn, generate code for the MIPS @scr@varnnnn chip. This tells the assembler to accept instructions specific to that chip, and to schedule for that chip's hazards.

@item -march=@varcpu Generate code for a particular MIPS cpu. It is exactly equivalent to @samp-m@varcpu, except that there are more value of @varcpu understood. Valid @varcpu value are:

@quotation 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, 4kc, 4km, 4kp, 4ksc, 4kec, 4kem, 4kep, 4ksd, m4k, m4kp, 24kc, $24kf2_1, 24kf, 24kf1_1, 24kec, 24kef2_1, 24kef, 24kef1_1, 34kc,$

For compatibility reasons, @samp@varnx and @samp@varbfx are accepted as synonyms for @samp@varnf1_1.$Theseval$

@item -mtune=@varcpu Schedule and tune for a particular MIPS cpu. Valid @varcpu values are identical to @samp-march=@varcpu.

@item -mabi=@varabi Record which ABI the source code uses. The recognized arguments are: @samp32, @sampn32, @sampo64, @samp64 and @sampeabi.

@item -msym32 @itemx -mno-sym32 @cindex -msym32 @cindex -mno-sym32 Equivalent to adding @code.set sym32 or @code.set nosym32 to the beginning of the assembler input. @xrefMIPS symbol sizes.

@cindex @code-nocpp ignored (MIPS) @item -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 @scgnu @code@valueAS, there is no need for @samp-nocpp, because the @scgnu assembler itself never runs the C preprocessor.

@item –construct-floats @itemx –no-construct-floats @cindex –construct-floats @cindex –no-construct-floats The @code–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 @code–construct-floats is selected, allowing construction of these floating point constants.

@item –trap @itemx –no-break @c FIXME! (1) reflect these options (next item too) in option summaries; @c (2) stop teasing, say $_whichinstructionsexpanded_how$ @*code*@*valueASautomaticallymacroexpandscertaindivisionand*

@item –break @itemx –no-trap Generate code to take a break exception rather than a trap exception when an error is detected. This is the default.

@item -mpdr @itemx -mno-pdr Control generation of @code.pdr sections. Off by default on IRIX, on elsewhere.

@item -mshared @itemx -mno-shared When generating code using the Unix calling conventions (selected by @samp-KPIC or @samp-mcall$_shared$), $gaswillnormallygeneratecodewhichcangointoasharedlibrary.The@samp-mn$ *ops*.@*endtable*

@node MIPS Object @section MIPS ECOFF object code

@cindex ECOFF sections @cindex MIPS ECOFF sections Assembling for a @scmips @scecoff target supports some additional sections besides the usual @code.text, @code.data and @code.bss. The additional sections are @code.rdata, used for read-only data, @code.sdata, used for small data, and @code.sbss, used for small common objects.

@cindex small objects, MIPS ECOFF @cindex @codegp register, MIPS When assembling for @scecoff, the assembler uses the @code*gp*(@*code*28) register to form the address of a "small object". Any object in the @code.sdata or @code.sbss sections is considered "small" in this sense. For external objects, or for objects in the @code.bss section, you can use the @code@valueGCC @samp-G option to control the size of objects addressed via @code*gp*; $thedefaultvalueis8, meaningthatareferencetoanyobjecteightbytesorsmalleruses$@*code*gp. Passing @samp-G 0 to @code@valueAS prevents it from using the @code$gpregisteronthebasisofobjectsize(buttheassembl$ for objects in @code.sdata or @codesbss in any case). The size of an object in the @code.bss section is set by the @code.comm or @code.lcomm directive that defines it. The size of an external object may be set with the @code.extern directive. For example, @samp.extern sym,4 declares that the object at @codesym is 4 bytes in length, whie leaving @codesym otherwise undefined.

Using small @scecoff objects requires linker support, and assumes that the @code$gpregisteriscorrectlyinitialized(nor$ register.

@node MIPS Stabs @section Directives for debugging information

@cindex MIPS debugging directives @scmips @scecoff @code@valueAS supports several directives used for generating debugging information which are not support by traditional @scmips assemblers. These are @code.def, @code.endef, @code.dim, @code.file, @code.scl, @code.size, @code.tag, @code.type, @code.val, @code.stabd, @code.stabn, and @code.stabs. The debugging information generated by the three @code.stab directives can only be read by @scgdb, not by traditional @scmips debuggers (this enhancement is required to fully support C++ debugging). These directives are primarily used by compilers, not assembly language programmers!

@node MIPS symbol sizes @section Directives to override the size of symbols

@cindex @code.set sym32 @cindex @code.set nosym32 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 @sampdla 4, $symisusually$:

@smallexample lui 4, lui1,daddiu 4,4,daddiu 1,1,dsll32 4,4,0 daddu 4,4,1@*endsmallexample*

whereas the 32-bit expansion is simply:

@smallexample lui 4, $daddiu$4,4, @*endsmallexample*

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 @code.set sym32 directive to tell the assembler that, from this point on, all expressions of the form @samp@varsymbol or @samp@varsymbol + @varoffset have 32-bit values. For example:

@smallexample .set sym32 dla 4, $symlw$4,sym+16 sw 4, $sym + 0x8000(4)$ @end smallexample

will cause the assembler to treat @sampsym, @codesym+16 and @codesym+0x8000 as 32-bit values. The handling of non-symbolic addresses is not affected.

The directive @code.set nosym32 ends a @code.set sym32 block and reverts to the normal behavior. It is also possible to change the symbol size using the command-line options @option-msym32 and @option-mno-sym32.

These options and directives are always accepted, but at present, they have no effect for anything other than n64.

@node MIPS ISA @section Directives to override the ISA level

@cindex MIPS ISA override @kindex @code.set mips@varn @scgnu @code@valueAS supports an additional directive to change the @scmips Instruction Set Architecture level on the fly: @code.set mips@varn. @varn should be a number from 0 to 5, or 32, 32r2, 64 or 64r2. The values other than 0 make the assembler accept instructions for the corresponding @scisa level, from that point on in the assembly. @code.set mips@varn affects not only which instructions are permitted, but also how certain macros are expanded. @code.set mips0 restores the @scisa 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 @scmips3 instructions while assembling in 32 bit mode. Use this directive with care!

@cindex MIPS CPU override @kindex @code.set arch=@varcpu The @code.set arch=@varcpu 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 @samp-march command line option are also selectable by this directive. The original value is restored by @code.set arch=default.

The directive @code.set mips16 puts the assembler into MIPS 16 mode, in which it will assemble instructions for the MIPS 16 processor. Use @code.set nomips16 to return to normal 32 bit mode.

Traditional @scmips assemblers do not support this directive.

@node MIPS autoextend @section Directives for extending MIPS 16 bit instructions

@kindex @code.set autoextend @kindex @code.set noautoextend By default, MIPS 16 instructions are automatically extended to 32 bits when necessary. The directive @code.set noautoextend will turn this off. When @code.set noautoextend is in effect, any 32 bit instruction must be explicitly extended with the @code.e modifier (e.g., @codeli.e 4, 1000).*Thedirective@code.setautoextendmaybeusedtooncea*

This directive is only meaningful when in MIPS 16 mode. Traditional @scmips assemblers do not support this directive.

@node MIPS insn @section Directive to mark data as an instruction

@kindex @code.insn The @code.insn directive tells @code@valueAS that the following data is actually instructions. This makes a difference in MIPS 16 mode: when loading the address of a label which precedes instructions, @code@valueAS automatically adds 1 to the value, so that jumping to the loaded address will do the right thing.

@node MIPS option stack @section Directives to save and restore options

@cindex MIPS option stack @kindex @code.set push @kindex @code.set pop The directives @code.set push and @code.set pop may be used to save and restore the current settings for all the options which are controlled by @code.set. The @code.set push directive saves the current settings on a stack. The @code.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 @scmips assemblers do not support these directives.

@node MIPS ASE instruction generation overrides @section Directives to control generation of MIPS ASE instructions

@cindex MIPS MIPS-3D instruction generation override @kindex @code.set mips3d @kindex @code.set nomips3d The directive @code.set mips3d makes the assembler accept instructions from the MIPS-3D Application Specific Extension from that point on in the assembly. The @code.set nomips3d directive prevents MIPS-3D instructions from being accepted.

@cindex SmartMIPS instruction generation override @kindex @code.set smartmips @kindex @code.set nosmartmips The directive @code.set smartmips makes the assembler accept instructions from the SmartMIPS Application Specific Extension to the MIPS32 @scisa from that point on in the assembly. The @code.set nosmartmips directive prevents SmartMIPS instructions from being accepted.

@cindex MIPS MDMX instruction generation override @kindex @code.set mdmx @kindex @code.set nomdmx The directive @code.set mdmx makes the assembler accept instructions from the MDMX Application Specific Extension from that point on in the assembly. The @code.set nomdmx directive prevents MDMX instructions from being accepted.

@cindex MIPS DSP Release 1 instruction generation override @kindex @code.set dsp @kindex @code.set

nodsp The directive @code.set dsp makes the assembler accept instructions from the DSP Release 1 Application Specific Extension from that point on in the assembly. The @code.set nodsp directive prevents DSP Release 1 instructions from being accepted.

@cindex MIPS DSP Release 2 instruction generation override @kindex @code.set dspr2 @kindex @code.set nodspr2 The directive @code.set dspr2 makes the assembler accept instructions from the DSP Release 2 Application Specific Extension from that point on in the assembly. This dirctive implies @code.set dsp. The @code.set nodspr2 directive prevents DSP Release 2 instructions from being accepted.

@cindex MIPS MT instruction generation override @kindex @code.set mt @kindex @code.set nomt The directive @code.set mt makes the assembler accept instructions from the MT Application Specific Extension from that point on in the assembly. The @code.set nomt directive prevents MT instructions from being accepted.

Traditional @scmips assemblers do not support these directives.