# Pseudo Op-Codes

This chapter describes pseudo op-codes (directives). These pseudo op-codes influence the assembler's later behavior. In the text, boldface type specifies a keyword and italics represents an operand that you define.

The assembler has the pseudo op-codes listed in Table 8-1.

**Table 8-1** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.2byte** *expression1 [ , expression2 ] ... [ , expressionN]** | Truncates the expressions in the comma-separated list to 16-bit values and assembles the values in successive locations. The *expression*s must be absolute or in the form of a label difference ( *label1* - *label2*) if both labels are defined in the same section.<br>This directive optionally can have the form *expression1* [ : *expression2* ]. The *expression2* replicates expression1's value *expression2* times.<br>This directive does no automatic alignment. (*64-bit and N32 only) |
| **.4byte** *expression1 [ , expression2 ] ... [ , expressionN]** | Truncates the expressions in the comma-separated list to 32-bit values and assembles the values in successive locations.The *expression*s must be absolute or in the form of a label difference ( *label1* - *label2*) if both labels are defined in the same section.<br>This directive optionally can have the form *expression1* [ : *expression2* ]. The *expression2* replicates expression1's value *expression2* times.<br>This directive does no automatic alignment. (*64-bit and N32 only) |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.8byte** *expression1 [ , expression2 ] ... [ , expressionN]** | Truncates the expressions in the comma-separated list to 64-bit values and assembles the values in successive locations. The *expression*s must be absolute or in the form of a label difference ( *label1 - label2*) if both labels are defined in the same section.<br>This directive optionally can have the form *expression1* [ : *expression2* ]. The *expression2* replicates expression1's value *expression2* times. This directive does no automatic alignment. (*64-bit and N32 only) |
| .**aent** *name***, symno** | Sets an alternate entry point for the current procedure. Use this information when you want to generate information for the debugger. It must appear inside an *.ent* / *.end* pair. |
| **.alias** *reg1, reg2** | Indicates that memory reference through the two registers (*reg1, reg2*) will overlap. The compiler uses this form to improve instruction scheduling. (32-bit only.) |
| **.align** *expression* | Advances the location counter to make the *expression* low order bits of the counter zero. Normally, the *.half, .word, .float*, and *.double* directives automatically align their data appropriately. For example, *.word* does an implicit *.align* 2 (*.double* does an *.align* 3). You disable the automatic alignment feature with *.align* 0. The assembler reinstates automatic alignment at the next *.text*, *.data*, *.rdata*, or *.sdata* directive.<br>Labels immediately preceding an automatic or explicit alignment are also realigned. For example, *foo: .align 3; .word 0* is the same as *.align 3; foo: .word 0*. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.ascii** *string [, string]...* | Assembles each *string* from the list into successive locations. The *.ascii* directive does not null pad the string. You MUST put quotation marks (") around each string. You can use the backslash escape characters. For a list of the backslash characters, see Chapter 4. |
| **.asciiz** *string [, string]...* | Assembles each *string* in the list into successive locations and adds a null. You can use the backslash escape characters. For a list of the backslash characters, see Chapter 4. |
| **.asm0*** | Tells the assembler's second pass that this assembly came from the first pass.For use by compilers) (*32-bit only.) |
| **.bgnb** *symno** | Sets the beginning of a language block. For use by compilers. The *.bgnb* and *.endb* directives delimit the scope of a variable set. The scope can be an entire procedure, or it can be a nested scope (for example a "{}" block in the C language). The symbol number *symno* refers to a dense number in a *.T* file. For an explanation of *.T* files, see the *MIPSpro Compiling, Debugging and Performance Tuning Guide*. To set the end of a language block, see *.endb*. (*32-bit only.) |
| **.byte** *expression1 [, expression2 ] ... [, expressionN]* | Truncates the expressions from the comma-separated list to 8-bit values, and assembles the values in successive locations. The *expression*s must be absolute. The operands can optionally have the form: *expression1* [ : *expression2* ]. The *expression2* replicates *expression1's* value *expression2* times. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.comm** *name, expression [alignment]* | Unless defined elsewhere, *name* becomes a global common symbol at the head of a block of *expression* bytes of storage. The linker overlays like-named common blocks, using the maximum of the *expression*s. The 64-bit and N32 assembler also accepts an optional value which specifies the alignment of the symbol. |
| **.cpadd** *reg* | Emits code that adds the value of "_gp" to *reg*. |
| **.cpload** *reg* | Expands into the three instructions function prologue that sets up the *$gp* register. This directive is used by position-independent code. |
| **.cplocal** *reg** | Causes the assembler to use *reg* instead of *$gp* as the context pointer. This directive is used by position-independent code.<br>(*64-bit and N32 only) |
| **.cprestore** *offset* | Causes the assembler to emit the following at the point where it occurs:<br>sw $gp, offset ($sp)<br>Also, causes the assembler to generate:<br>lw $gp, offset ($sp)<br>after every JAL or BAL operation. Offset should point to the saved register area as described in Chapter 7.<br>This directive is used by position-independent code following the caller saved gp convention. |
| **.cpreturn*** | Causes the assembler to emit the following at the point where it occurs:<br>ld $gp, offset ($sp)<br>The *offset* is obtained from the previous **.cpsetup** pseudo-op.<br>(*64-bit and N32 only) |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.cpsetup** *reg1, {offset \| reg2}, label** | Causes the assembler to emit the following at the point where it occurs:<br>sd $gp, offset ($sp)<br>lui $gp, 0 { label }<br>daddiu $gp, $gp, 0 { label }<br>daddu $gp, $gp, reg1<br>ld $gp, offset ($sp)<br>This sequence is used by position-independent code following the callee saved gp convention. It stores *$gp* in the saved register area and calculates the virtual address of *labe*l and places it in *reg1*. By convention, *reg1* is *$25* (*t9*).<br>If *reg2* is used instead of offset, *$gp* is saved and restored to and from this register.<br>(*64-bit and N32 only) |
| **.data** | Tells the assembler to add all subsequent data to the data section. |
| .**double** *expression [ , expression2] ...[, expressionN]* | Initializes memory to 64-bit floating point numbers. The operands optionally can have the form: *expression1* [ : *expression2* ]. The *expression1* is the floating point value. The optional *expression2* is a non-negative expression that specifies a repetition count. The *expression2* replicates *expression1*'s value *expression2* times. This directive aligns its data and any preceding labels automatically to a double-word boundary. You can disable this feature by using *.align* 0. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.dword** *expression [ , expression2 ] ...[, expressionN]* | Truncates the expressions in the comma-separated list to 64-bits and assembles the values in successive locations. The expressions must be absolute. The operands optionally can have the form: *expression1* [:*expression2*]. The *expresssion2* replicates *expression1*'s value *expression2* number of times. The directive aligns its data and preceding labels automatically to a doubleword boundary. You can disable this feature by using *.align* 0. |
| **.end** *[proc_name]* | Sets the end of a procedure. Use this directive when you want to generate information for the debugger. To set the beginning of a procedure, see *.ent*. |
| **.endb** *symno** | Sets the end of a language block. To set the beginning of a language block, see *.bgnb*. (*32-bit only.) |
| .**endr** | Signals the end of a repeat block. To start a repeat block, see *.repeat*. |
| **.ent** *proc_name* | Sets the beginning of the procedure *proc_name*. Use this directive when you want to generate information for the debugger. To set the end of a procedure, see *.end*. |
| **.extern** *name expression* | *name* is a global undefined symbol whose size is assumed to be *expression* bytes. The advantage of using this directive, instead of permitting an undefined symbol to become global by default, is that the assembler can decide whether to use the economical *$gp*-relative addressing mode, depending on the value of the –**G** option. As a special case, if *expression* is zero, the assembler refrains from using *$gp* to address this symbol regardless of the size specified by –**G**. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| .**err*** | Signals an error. For use by compilers. Any compiler front-end that detects an error condition puts this directive in the input stream. When the assembler encounters a *.err*, it quietly ceases to assemble the source file. This prevents the assembler from continuing to process a program that is incorrect. (*32-bit only.) |
| **.file** *file_number file_name_string* | Specifies the source file corresponding to the assembly instructions that follow. For use only by compilers, not by programmers; when the assembler sees this, it refrains from generating line numbers for *dbx* to use unless it also sees *.loc* directives. |
| **.float** *expression1 [ , expression2 ] ... [, expressionN]* | Initializes memory to single precision 32-bit floating point numbers. The operands optionally can have the form: *expression1* [ : *expression2* ]. The optional *expression2* is a non-negative expression that specifies a repetition count. This optional form replicates *expression1*'s value *expression2* times. This directive aligns its data and preceding labels automatically to a word boundary. You can disable this feature by using *.align* 0. |
| **.fmask** *mask offset* | Sets a mask with a bit turned on for each floating point register that the current routine saved. The least-significant bit corresponds to register *$f0*. The offset is the distance in bytes from the virtual frame pointer at which the floating point registers are saved. The assembler saves higher register numbers closer to the virtual frame pointer. You must use *.ent* before *.fmask* and only one *.fmask* may be used per *.ent*. Space should be allocated for those registers specified in the *.fmask*. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.frame** *frame-register offset return_pc_register* | Describes a stack frame. The first register is the frame-register, the offset is the distance from the frame register to the virtual frame pointer, and the second register is the return program counter (or, if the first register is *$0*, this directive shows that the return program counter is saved four bytes from the virtual frame pointer). You must use *.ent* before *.frame* and only one *.frame* may be used per *.ent*. No stack traces can be done in the debugger without *.frame*. |
| **.globl** *name* | Makes the *name* external. If the name is defined otherwise (by its appearance as a label), the assembler will export the symbol; otherwise it will import the symbol. In general, the assembler imports undefined symbols (that is, it gives them the UNIX storage class “global undefined” and requires the linker to resolve them). |
| **.gjaldef** *int_bitmask fp_bitmask** | Sets the masks defining the registers whose value is preserved during a procedure call. For use by compilers. See Table 1-1 for the default for integer saved registers. (*32-bit only.) |
| **.gjallive** *int_bitmask fp_bitmask** | Sets the default masks for live registers before a procedure call (A JAL instruction). For use by compilers. (*32-bit only.) |
| **.gjrlive** *int_bitmask fp_bitmask** | Sets the default masks for live registers before a procedure’s return (A JR instruction). For use by compilers. (*32-bit only.) |
| **.gpword** *local-sym* | This directive is similar to *.word* except that the relocation entry for local-sym has the R_MIPS_GPREL32 type. After linkage, this results in a 32-bit value that is the distance between *local*-sym and *gp*. *local-sym* must be local. This directive is used by the code generator for PIC switch tables. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.half** *expression1 [ , expression2 ] ... {, expressionN]* | Truncates the expressions in the comma-separated list to 16-bit values and assembles the values in successive locations. The *expression*s must be absolute. This directive optionally can have the form: *expression1* [ : *expression2* ]. The *expression2* replicates expression1's value *expression2* times. This directive automatically aligns its data appropriately. You can disable this feature by using *.align* 0. |
| **.lab** *label_name* | Associates a named label with the current location in the program text. For use by compilers. |
| **.lcomm** *name, expression* | Makes the *name*'s data type *bss*. The assembler allocates the named symbol to the *bss* area, and the expression defines the named symbol's length. If a *.globl* directive also specifies the name, the assembler allocates the named symbol to external *bss*. The assembler puts *bss* symbols in one of two *bss* areas. If the defined size is smaller than (or equal to) the size specified by the assembler or compiler's **–G** command line option, the assembler puts the symbols in the *sbss* area and uses *$gp* to address the data. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.livereg** *int_bitmask fp_bitmask** | Affects the next jump instruction even if it is not the successive instruction. For use by compilers. The *.livereg* directive may come before any of the following instructions: JAL, JR, and SYSCALL. By default, external J instructions and JR instructions through a register other than *$ra*, are treated as external calls; that is; all registers are assumed live. The directive *.livereg* cannot appear before an external J (it will affect the next JR, JAL, or SYSCALL instead of the J instruction). *.livereg* may appear before a JR instruction through a register other than *$ra*. The directive can't be used before a BREAK instruction. For BREAK instructions, the assembler also assumes all registers are live.<br><br>*.livereg* notes to the assembler which registers are live before a jump, in order to avoid unsafe optimizations by the reorganizer. The directive *.livereg* takes two arguments, *int_bitmask*, and *fp_bitmask*, which are 32 bit bitmasks with a bit turned on for each register that is live before a jump. The most significant bit corresponds to register *$0* (which is opposite to that used in other assembly directives, *.mask*, *.fmask*). The first bitmap indicates live integer registers and the second indicates live FPs. (*32-bit only) |
| **.loc** *file_number line_number [column]* | Specifies the source file and the line within that file that corresponds to the assembly instructions that follow. For use by compilers. The assembler ignores the file number when this directive appears in the assembly source file. Then, the assembler assumes that the directive refers to the most recent *.file* directive. The 64-bit and N32 assembler also supports an optional value that specifies the column number. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.mask** *mask, offset* | Sets a mask with a bit turned on for each general purpose register that the current routine saved. For use by compilers. Bit one corresponds to register *$1*. The offset is the distance in bytes from the virtual frame pointer where the registers are saved. The assembler saves higher register numbers closer to the virtual frame pointer. Space should be allocated for those registers appearing in the mask. If bit zero is set it is assumed that space is allocated for all 31 registers regardless of whether they appear in the mask. |
| **nada*** | Tells the assembler to put in an instruction that has no effect on the machine state. It has the same effect as *nop* (described below), but it produces more efficient code on an R8000. (*64-bit and N32 only) |
| **.noalias** *reg1, reg2** | Register1 and register2, when used as indexed registers to memory will never point to the same memory. The assembler will use this as a hint to make more liberal assumptions about resource dependency in the program. To disable this assumption, see *.alias.* (*32-bit only.) |
| **nop** | Tells the assembler to put in an instruction that has no effect on the machine state. While several instructions cause no-operation, the assembler only considers the ones generated by the nop directive to be wait instructions. This directive puts an explicit delay in the instruction stream.<br><br>**Note:** Unless you use ".set noreorder", the reorganizer may eliminate unnecessary "nop" instructions. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.option** *options* | Tells the assembler that certain options were in effect during compilation. (These options can, for example, limit the assembler's freedom to perform branch optimizations.) This option is intended for compiler-generated *.s* files rather than for hand-coded ones. |
| **.origin** *expression** | Specifiess the current offset in a section to the value of *expression*. (*64-bit and N32 only) |
| **.repeat** *expression* | Repeats all instructions or data between the *.repeat* directive and the *.endr* directive. The *expression* defines how many times the data repeats. With the *.repeat* directive, you cannot use labels, branch instructions, or values that require relocation in the block. To end a *.repeat*, see *.endr*. |
| **.rdata** | Tells the assembler to add subsequent data into the *rdata* section. |
| **.sdata** | Tells the assembler to add subsequent data to the *sdata* section. |
| **.section** *name [, section type, section flags, section entry size, section alignment]** | Instructs the assembler to create a section with the given name and optional attributes.<br>Legal *section type* values are denoted by variables prefixed by SHT_ in <*elf.h*>.<br>Legal *section flags* values are denoted by variables prefixed by SHF_ in <*elf.h*>.<br>The *section entry size* specifies the size of each entry in the section. For example, it is 4 for .text sections.<br>The *section alignment* specifies the byte boundary requirement for the section. For example, it is 16 for .text sections.<br>(*64-bit and N32 only) |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.set** *option* | Instructs the assembler to enable or to disable certain options. Use *.set* options only for hand-crafted assembly routines. The assembler has these default options: *reorder*, *macro*, and *at*. You can specify only one option for each *.set* directive. You can specify these *.set* options: \<br><br>The *reorder* option lets the assembler reorder machine language instructions to improve performance. The *noreorder* option prevents the assembler from reordering machine language instructions. If a machine language instruction violates the hardware pipeline constraints, the assembler issues a warning message.<br><br>The *bopt/nobopt* option lets the assembler perform branch optimization. This involves moving an instruction that is the target of a branch or jump instruction into the delay slot; this is performed only if no unpredictable side effects can occur.<br><br>The *macro* option lets the assembler generate multiple machine instructions from a single assembler instruction.<br><br>The *nomacro* option causes the assembler to print a warning whenever an assembler operation generates more than one machine language instruction. You must select the *noreorder* option before using the *nomacro* option; otherwise, an error results. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.set** *option* (continued) | The *at* option lets the assembler use the *$at* register for macros, but generates warnings if the source program uses *$at*. When you use the *noat* option and an assembler operation requires the *$at* register, the assembler issues a warning message; however, the *noat* option does let source programs use *$at* without issuing warnings.<br><br>The *nomove* option tells the assembler to mark each subsequent instruction so that it cannot be moved during reorganization. Because the assembler can still insert *nop* instructions where necessary for pipeline constraints, this option is less stringent than *noreorder*. The assembler can still move instructions from below the *nomove* region to fill delay slots above the region or vice versa. The *nomove* option has part of the effect of the "volatile" C declaration; it prevents otherwise independent loads or stores from occurring in a different order than intended.<br><br>The *move* option cancels the effect of *nomove*.<br><br>The *notransform* option tells the assembler to mark each subsequent instruction so that it cannot be transformed by *pixie*(1), into an equivalent set of instructions. For an overview of *pixie*(1) see the *MIPSpro Compiling, Debugging, and Performance Tuning Guide*.<br><br>The *transform* option cancels the effect of *notransform*. |
| **.size** *name, expression* | Specifiess the size of an object denoted by *name* to the value of *expression*. |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.space** *expression* | Advances the location counter by the value of the specified *expression* bytes. The assembler fills the space with zeros. |
| **.struct** *expression* | This permits you to lay out a structure using labels plus directives like *.word*, *.byte*, and so forth. It ends at the next segment directive (*.data*, *.text*, etc.). It does not emit any code or data, but defines the labels within it to have values which are the sum of *expression* plus their offsets from the *.struct* itself. |
| (*symbolic equate*) | Takes one of these forms: *name* = *expression* or *name* = *register*. You must define the name only once in the assembly, and you **cannot** redefine the name. The expression must be computable when you assemble the program, and the expression must involve operators, constants, and equated symbols. You can use the name as a constant in any later statement. |
| **.text** | Tells the assembler to add subsequent code to the *text* section. (This is the default.) |
| **.type** *name, value** | Specifies the elf type of an object denoted by name to value. Legal elf type values are denoted by variables prefixed by STT_ in <*elf.h*>. (*64-bit and N32 only) |
| **.verstamp** *major minor* | Specifies the major and minor version numbers (for example, version 0.15 would be *.verstamp* 0 15). |
| **.vreg** *register offset symno** | Describes a register variable by giving the offset from the virtual frame pointer and the symbol number *symno* (the dense number) of the surrounding procedure. For use by compilers. (*32-bit only.) |

**Table 8-1 (continued)** Pseudo Op-Codes

| Pseudo-Op | Description |
|---|---|
| **.weakext** *weak_name [strong_name]* | Defines a weak external name and optionally associates it with the *strong_name*. |
| .**word** *expression1 [, expression2 ] ... [, expressionN]* | Truncates the expressions in the comma-separated list to 32-bits and assembles the values in successive locations. The expressions must be absolute. The operands optionally can have the form: *expression1* [ : *expression2* ]. The *expression2* replicates *expression1*'s value *expression2* times. This directive aligns its data and preceding labels automatically to a word boundary. You can disable this feature by using *.align* 0. |