Pseudo-Operations A

The pseudo-operations listed in this appendix are supported by the SPARC assembler.

Alphabetized Listing with Descriptions

`.alias`

Turns off the effect of the preceding `.noalias` pseudo-op. (Compiler-generated only.)

`.align` *`boundary`*

Aligns the location counter on a boundary where `((“location counter” mod boundary)==0)`; *boundary* may be any power of 2.

`.ascii` *`string`* `[,` *`string`*`"]`

Generates the given sequence(s) of ASCII characters.

`.asciz` *`string`* `[,` *`string`*`]*`

Generates the given sequence(s) of ASCII characters. This pseudo-op appends a null (zero) byte to each *string*.

`.byte` *`8bitval`* `[,` *`8bitval`*`]*`

Generates (a sequence of) initialized bytes in the current segment.

`.common` *`symbol`*`,` *`size`* `[,` *`sect_name`*`]` `[,` *`alignment`*`]`

Provides a tentative definition of *symbol*. *Size* bytes are allocated for the object represented by *symbol*.

- If the symbol is not defined in the input file and is declared to be *local* to the file, the symbol is allocated in *sect_name* and its location is optionally aligned to a multiple of *alignment*. If *sect_name* is not given, the symbol is allocated in the uninitialized data section (*bss*). Currently, only `.bss` is supported for the section name. (`.data` is not currently supported.)
- If the symbol is not defined in the input file and is declared to be *global*, the SPARC link editor allocates storage for the symbol, depending on the definition of *symbol_name* in other files. Global is the default binding for common symbols.
- If the symbol is defined in the input file, the definition specifies the location of the symbol and the tentative definition is overridden.

`.double` `0r`*`floatval`* `[, 0r`*`floatval`*`]*`

Generates (a sequence of) initialized double-precision floating-point values in the current segment. *floatval* is a string acceptable to atof(3); that is, an optional sign followed by a non-empty string of digits with optional decimal point and optional exponent.

`.empty`

Suppresses assembler complaints about the next instruction presence in a delay slot when used in the delay slot of a Control Transfer Instruction (CTI).

Some instructions should not be in the delay slot of a CTI. See the S*PARC Architecture Manual* for details.

`.file` *`string`*

Creates a symbol table entry where *string* is the symbol name and `STT_FILE` is the symbol table type. *string* specifies the name of the source file associated with the object file.

`.global` *`symbol`* [, *`symbol`*]*
`.globl` *`symbol`* [, *`symbol`*]*

Declares each *symbol* in the list to be global; that is, each symbol is either defined externally or defined in the input file and accessible in other files; default bindings for the symbol are overridden.

- A global symbol definition in one file will satisfy an undefined reference to the same global symbol in another file.
- Multiple definitions of a defined global symbol is not allowed. If a defined global symbol has more than one definition, an error will occur.
- A global psuedo-op oes not need to occur before a definition, or tentative definition, of the specified symbol.

Note – This pseudo-op by itself does not define the symbol.

`.half` *`16bitval`* [, *`16bitval`*]*

Generates (a sequence of) initialized halfwords in the current segment. The location counter must already be aligned on a halfword boundary (use `.align 2`).

`.ident` *`string`*

Generates the null terminated string in a comment section. This operation is equivalent to:

```
.pushsection .comment
.asciz string
.popsection
```

`.local` *`symbol`* [, *`symbol`*]*

Declares each *symbol* in the list to be local; that is, each symbol is defined in the input file and not accessible in other files; default bindings for the symbol are overridden. These symbols take precedence over *weak* and *global* symbols.

Since local symbols are not accessible to other files, local symbols of the same name may exist in multiple files.

Note – This pseudo-op by itself does not define the symbol.

`.noalias` *`%reg1, %reg2`*

%reg1 and *%reg2* will not alias each other (that is, point to the same destination) until a `.alias` pseudo-op is issued. (Compiler-generated only.)

`.nonvolatile`

Defines the end of a block of instruction. The instructions in the block may not be permuted. This pseudo-op has no effect if:

- The block of instruction has been previously terminated by a Control Transfer Instruction (CTI) or a label
- There is no preceding `.volatile` pseudo-op

`.optim` *`string`*

This pseudo-op changes the optimization level of a particular function. (Compiler-generated only.)

`.popsection`

Removes the top section from the section stack. The new section on the top of the stack becomes the current section. This pseudo-op and its corresponding `.pushsection` command allow you to switch back and forth between the named sections.

`.proc` *`n`*

Signals the beginning of a *procedure* (that is, a unit of optimization) to the peephole optimizer in the SPARC assembler; *n* specifies which registers will contain the return value upon return from the procedure. (Compiler-generated only.)

`.pushsection` *`sect_name`* `[,` *`attributes`*`]`

Moves the named section to the top of the section stack. This new top section then becomes the current section. This pseudo-op and its corresponding `.popsection` command allow you to switch back and forth between the named sections.

`.quad 0r`*`floatval`* `[, 0r`*`floatval`*`]*`

Generates (a sequence of) initialized quad-precision floating-point values in the current segment. *floatval* is a string acceptable to `atof(3)`; that is, an optional sign followed by a non-empty string of digits with optional decimal point and optional exponent.

Note – The `.quad` command currently generates quad-precision values with only double-precision significance.

`.reserve` *`symbol`*`,` *`size`* `[,` *`sect_name`* `[,` *`alignment`*`]]`

Defines *symbol*, and reserves *size* bytes of space for it in the *sect_name*. This operation is equivalent to:

```
    .pushsection    sect_name
    .align          alignment
symbol:
    .skip           size
    .popsection
```

If a section is not specified, space is reserved in the current segment.

`.section` *`section_name`* `[,` *`attributes`*`]`

Makes the specified section the current section.

The assembler maintains a section stack which is manipulated by the section control directives. The current section is the section that is currently on top of the stack. This pseudo-op changes the top of the section stack.

- If *section_name* does not exist, a new section with the specified name and attributes is created.

- If *section_name* is a non-reserved section, *attributes* must be included the first time it is specified by the `.section` directive.

See the sections "Predefined User Sections" and "Predefined Non-User Sections" in Chapter 3, "Executable and Linking Format," for a detailed description of the reserved sections. See Table 3-2 in Chapter 3, "Executable and Linking Format," for a detailed description of the section attribute flags.

Attributes can be:

```
#write | #alloc | #execinstr
```

`.seg` *`section_name`*

Note – This pseudo-op is currently supported for compatibility with existing SunOS 4.1 SPARC assembly language programs. This pseudo-op has been replaced by the `.section` pseudo-op.

Changes the current section to one of the predefined user sections. The assembler will interpret the following SunOS 4.1 SPARC assembly directive:

```
.seg text, .seg data, .seg data1, .seg bss,
```

to be the same as the following SunOS 5.*x* SPARC assembly directive:

```
.section .text, .section .data, .section .data1,
.section .bss.
```

Predefined user section names are changed in SunOS 5.*x*.

`.single 0r`*`floatval`* `[, 0r`*`floatval`*`]*`

Generates (a sequence of) initialized single-precision floating-point values in the current segment.

Note – This operation does not align automatically.

`.size  symbol,  expr`

Declares the symbol size to be *expr*. *expr* must be an absolute expression.

`.skip  n`

Increments the location counter by *n*, which allocates *n* bytes of empty space in the current segment.

`.stabn <various parameters>`

The pseudo-op is used by Solaris 2.*x* SPARCompilers only to pass debugging information to the symbolic debuggers.

`.stabs <various parameters>`

The pseudo-op is used by Solaris 2.*x* SPARCompilers only to pass debugging information to the symbolic debuggers.

`.type  symbol,  type`

Declares the type of symbol, where *type* can be:

```
#object
#function
#no_type
```

See Table 3-6 in Chapter 3, "Executable and Linking Format," for detailed information on symbols.

`.uahalf  16bitval  [, 16bitval]*`

Generates a (sequence of) 16-bit value(s).

Note – This operation does not align automatically.

`.uaword  32bitval  [, 32bitval]*`

Generates a (sequence of) 32-bit value(s).

Note – This operation does not align automatically.

`.version` *`string`*

Identifies the minimum assembler version necessary to assemble the input file. You can use this pseudo-op to ensure assembler-compiler compatibility. If *string* indicates a newer version of the assembler than this version of the assembler, a fatal error message is displayed and the SPARC assembler exits.

`.volatile`

Defines the beginning of a block of instruction. The instructions in the section may not be changed. The block of instruction should end at a `.nonvolatile` pseudo-op and should not contain any Control Transfer Instructions (CTI) or labels. The volatile block of instructions is terminated after the last instruction preceding a CTI or label.

`.weak` *`symbol`* `[,` *`symbol`*`]`

Declares each *symbol* in the list to be defined either externally, or in the input file and accessible to other files; default bindings of the symbol are overridden by this directive.

Note the following:

- A *weak* symbol definition in one file will satisfy an undefined reference to a global symbol of the same name in another file.
- Unresolved *weak* symbols have a default value of zero; the link editor does not resolve these symbols.
- If a *weak* symbol has the same name as a defined *global* symbol, the weak symbol is ignored and no error results.

Note – This pseudo-op does not itself define the symbol.

`.word` *`32bitval`* `[,` *`32bitval`*`]*`

Generates (a sequence of) initialized words in the current segment.

Note – This operation does not align automatically.

`.xstabs <`*`various parameters`*`>`

The pseudo-op is used by Solaris 2.*x* SPARCompilers only to pass debugging information to the symbolic debuggers.

`symbol =expr`

Assigns the value of *expr* to *symbol*.

A