Comments

Two types of comments are accepted by most SPARC assemblers: C-style "`/* . . . */`" comments (which may span multiple lines), and "`! . . .`" comments, which extend from the "`!`" to the end of the line.

A.2. Syntax Design

The suggested SPARC assembly language syntax is designed so that:

- The destination operand (if any) is consistently specified as the last (right-most) operand in an assembly language statement.
- A reference to the **contents** of a memory location (in a Load, Store, or SWAP instruction) is always indicated by square brackets ([ ]). A reference to the **address** of a memory location (such as in a JMPL, CALL, or SETHI) is specified directly, without square brackets.

A.3. Synthetic Instructions

The table shown below describes the mapping of a set of synthetic (or "pseudo") instructions to actual SPARC instructions. These synthetic instructions may be provided in a SPARC assembler for the convenience of assembly language programmers.

Note that synthetic instructions should not be confused with "pseudo-ops", which typically provide information to the assembler but do not generate instructions. Synthetic instructions always generate instructions; they provide more mnemonic syntax for standard SPARC instructions.

Table A-1 *Mapping of Synthetic Instructions to SPARC Instructions*

Synthetic Instruction	***SPARC Instruction(s)***	***Comment***
`cmp` *reg*$_{rs1}$`,` *reg_or_imm*	`subcc` *reg*$_{rs1}$`,` *reg_or_imm*`,%g0`	*compare*
`jmp` *address*	`jmpl` *address*`,%g0`	
`call` *address*	`jmpl` *address*`,%o7`	
`tst` *reg*$_{rs2}$	`orcc %g0,` *reg*$_{rs2}$`,%g0`	*test*
`ret`	`jmpl %i7+8,%g0`	*return from subroutine*
`retl`	`jmpl %o7+8,%g0`	*return from leaf subroutine*
`restore`	`restore %g0,%g0,%g0`	*trivial* `restore`
`save`	`save %g0,%g0,%g0`	*trivial* `save` *(Warning: trivial* `save` *should only be used in kernel code!)*
`set` *value*`,` *reg*$_{rd}$	`sethi %hi(`*value*`),` *reg*$_{rd}$ **or** `or %g0,` *value*`,` *reg*$_{rd}$ **or** `sethi %hi(`*value*`),` *reg*$_{rd}$`;` `or` *reg*$_{rd}$`,%lo(`*value*`),` *reg*$_{rd}$	*(when ((value&0x1fff) == 0))* *(when -4096 ≤ value ≤ 4095)* *(otherwise)* *Warning: do not use* `set` *in the delay slot of a DCTI.*
`not` *reg*$_{rs1}$`,` *reg*$_{rd}$	`xnor` *reg*$_{rs1}$`,%g0,` *reg*$_{rd}$	*one's complement*
`not` *reg*$_{rd}$	`xnor` *reg*$_{rd}$`,%g0,` *reg*$_{rd}$	*one's complement*
`neg` *reg*$_{rs2}$`,` *reg*$_{rd}$	`sub %g0,` *reg*$_{rs2}$`,` *reg*$_{rd}$	*two's complement*
`neg` *reg*$_{rd}$	`sub %g0,` *reg*$_{rd}$`,` *reg*$_{rd}$	*two's complement*

Table A-1 *Mapping of Synthetic Instructions to SPARC Instructions— Continued*

Synthetic Instruction		*SPARC Instruction(s)*		*Comment*
`inc`	*reg*$_{rd}$	`add`	*reg*$_{rd}$`,1,`*reg*$_{rd}$	*increment by 1*
`inc`	*const13*`,`*reg*$_{rd}$	`add`	*reg*$_{rd}$`,`*const13*`,`*reg*$_{rd}$	*increment by const13*
`inccc`	*reg*$_{rd}$	`addcc`	*reg*$_{rd}$`,1,`*reg*$_{rd}$	*increment by 1 and set icc*
`inccc`	*const13*`,`*reg*$_{rd}$	`addcc`	*reg*$_{rd}$`,`*const13*`,`*reg*$_{rd}$	*increment by const13 and set icc*
`dec`	*reg*$_{rd}$	`sub`	*reg*$_{rd}$`,1,`*reg*$_{rd}$	*decrement by 1*
`dec`	*const13*`,`*reg*$_{rd}$	`sub`	*reg*$_{rd}$`,`*const13*`,`*reg*$_{rd}$	*decrement by const13*
`deccc`	*reg*$_{rd}$	`subcc`	*reg*$_{rd}$`,1,`*reg*$_{rd}$	*decrement by 1 and set icc*
`deccc`	*const13*`,`*reg*$_{rd}$	`subcc`	*reg*$_{rd}$`,`*const13*`,`*reg*$_{rd}$	*decrement by const13 and set icc*
`btst`	*reg_or_imm*`,`*reg*$_{rs1}$	`andcc`	*reg*$_{rs1}$`,`*reg_or_imm*`,%g0`	*bit test*
`bset`	*reg_or_imm*`,`*reg*$_{rd}$	`or`	*reg*$_{rd}$`,`*reg_or_imm*`,`*reg*$_{rd}$	*bit set*
`bclr`	*reg_or_imm*`,`*reg*$_{rd}$	`andn`	*reg*$_{rd}$`,`*reg_or_imm*`,`*reg*$_{rd}$	*bit clear*
`btog`	*reg_or_imm*`,`*reg*$_{rd}$	`xor`	*reg*$_{rd}$`,`*reg_or_imm*`,`*reg*$_{rd}$	*bit toggle*
`clr`	*reg*$_{rd}$	`or`	`%g0,%g0,`*reg*$_{rd}$	*clear(zero) register*
`clrb`	[*address*]	`stb`	`%g0,`[*address*]	*clear byte*
`clrh`	[*address*]	`sth`	`%g0,`[*address*]	*clear halfword*
`clr`	[*address*]	`st`	`%g0,`[*address*]	*clear word*
`mov`	*reg_or_imm*`,`*reg*$_{rd}$	`or`	`%g0,`*reg_or_imm*`,`*reg*$_{rd}$	
`mov`	`%y,`*reg*$_{rd}$	`rd`	`%y,`*reg*$_{rd}$	
`mov`	`%asr`***n***`,`*reg*$_{rd}$	`rd`	`%asr`***n***`,`*reg*$_{rd}$	
`mov`	`%psr,`*reg*$_{rd}$	`rd`	`%psr,`*reg*$_{rd}$	
`mov`	`%wim,`*reg*$_{rd}$	`rd`	`%wim,`*reg*$_{rd}$	
`mov`	`%tbr,`*reg*$_{rd}$	`rd`	`%tbr,`*reg*$_{rd}$	
`mov`	*reg_or_imm*`,%y`	`wr`	`%g0,`*reg_or_imm*`,%y`	
`mov`	*reg_or_imm*`,%asr`***n***	`wr`	`%g0,`*reg_or_imm*`,%asr`***n***	
`mov`	*reg_or_imm*`,%psr`	`wr`	`%g0,`*reg_or_imm*`,%psr`	
`mov`	*reg_or_imm*`,%wim`	`wr`	`%g0,`*reg_or_imm*`,%wim`	
`mov`	*reg_or_imm*`,%tbr`	`wr`	`%g0,`*reg_or_imm*`,%tbr`	