Instruction-Set Mapping 5

The tables in this chapter describe the relationship between hardware instructions of the SPARC architecture, as defined in *The SPARC Architecture Manual* and the assembly language instruction set recognized by the SunOS 5.*x* SPARC assembler.

The SPARC-V9 instruction set is described in Appendix E, "SPARC-V9 Instruction Set."

Table Notation

Table 5-1 shows the table notation used in this chapter to describe the instruction set of the assembler. The following notations are commonly suffixed to assembler mnemonics (uppercase letters refer to SPARC architecture instruction names.

Table 5-1 Notations Used to Describe Instruction Sets

Notations	**Describes**	**Comment**
address	reg_{rs1} + reg_{rs2} reg_{rs1} + const13 reg_{rs1} − const13 const13 + reg_{rs1} const13	Address formed from register contents, immediate constant, or both.
asi		Alternate address space identifier; an unsigned 8-bit value. It can be the result of the evaluation of a symbol expression.
const13		A signed constant which fits in 13 bits. It can be the result of the evaluation of a symbol expression.
const22		A constant which fits in 22 bits. It can be the result of the evaluation of a symbol expression.
creg	%c0 ... %c31	Coprocessor registers.
freg	%f0 ... %f31	Floating-point registers.
imm7		A signed or unsigned constant that can be represented in 7 bits (it is in the range -64 ... 127). It can be the result of the evaluation of a symbol expression.
reg	%r0 ... %r31 %g0 ... %g7 %o0 ... %o7 %l0 ... %l7 %i0 ... %i7	General purpose registers. Same as %r0 ... %r7 (Globals) Same as %r8 ... %r15 (Outs) Same as %r16 ... %r23 (Locals) Same as %r24 ... %r31 (Ins)
reg_{rd}		Destination register.
reg_{rs1}, reg_{rs2}		Source register 1, source register 2.
reg_or_imm	reg_{rs2}, const13	Value from either a single register, or an immediate constant.

Table 5-1 Notations Used to Describe Instruction Sets *(Continued)*

Notations	Describes	Comment
regaddr	reg_{rs1} reg_{rs1} + reg_{rs2}	Address formed with register contents only.
Software_trap_ number	reg_{rs1} + reg_{rs2} reg_{rs1} + imm7 reg_{rs1} – imm7 uimm7 imm7 + reg_{rs1}	A value formed from register contents, immediate constant, or both. The resulting value must be in the range 0.....127, inclusive.
uimm7		An unsigned constant that can be represented in 7 bits (it is in the range 0 ... 127). It can be the result of the evaluation of a symbol expression.

Integer Instructions

The notations described in Table 5-2 are commonly suffixed to assembler mnemonics (uppercase letters for architecture instruction names).

Table 5-2 Commonly Suffixed Notations

Notation	Description
a	Instructions that deal with alternate space
b	Byte instructions
c	Reference to coprocessor registers
d	Doubleword instructions
f	Reference to floating-point registers
h	Halfword instructions
q	Quadword instructions
sr	Status register

Table 5-3 outlines the correspondence between SPARC hardware integer instructions and SPARC assembly language instructions.

The syntax of individual instructions is designed so that a destination operand (if any), which may be either a register or a reference to a memory location, is always the last operand in a statement.

Note – In Table 5-3,

- Braces ({ }) indicate optional arguments.
  Braces are not literally coded.
- Brackets ([ ]) indicate indirection: the contents of the addressed memory location are being read from or written to.
  Brackets are coded literally in the assembly language.
  Note that the usage of brackets described in Chapter 2, "Assembler Syntax" differs from the usage of these brackets.
- All `Bicc` and `Bfcc` instructions described may indicate that the annul bit is to be set by appending `",a"` to the opcode mnemonic; for example,

```
"bgeu,a label"
```

Table 5-3 SPARC to Assembly Language Mapping

Opcode	Mnemonic	Argument List	Operation	Comments
ADD ADDcc ADDX ADDXcc	add addcc addx addxcc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Add Add and modify `icc` Add with carry	
AND ANDcc ANDN ANDNcc	and andcc andn andncc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	And	

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
BN	bn{,a}	label	Branch on integer condition codes	branch never
BNE	bne{,a}	label		synonym: bnz
BE	be{,a}	label		synonym: bz
BG	bg{,a}	label		
BLE	ble{,a}	label		
BGE	bge{,a}	label		
BI	bl{,a}	label		
BGU	bgu{,a}	label		
BLEU	bleu{,a}	label		
BCC	bcc{,a}	label		synonym: bgeu
BCS	bcs{,a}	label		synonym: blu
BPOS	bpos{,a}	label		
BNEG	bneg{,a}	label		
BVC	bvc{,a}	label		
BVS	bvs{,a}	label		
BA	ba{,a}	label		synonym: b
CALL	call	label	Call subprogram	
CBccc	cbn{,a}	label	Branch on coprocessor condition codes	branch never
	cb3{,a}	label		
	cb2{,a}	label		
	cb23{,a}	label		
	cb1{,a}	label		
	cb13{,eo}	label		
	cb12{,a}	label		
	cb123{,a}	label		
	cb0{,a}	label		
	cb03{,a}	label		
	cb02{,a}	label		
	cb023{,a}	label		
	cb01{,a}	label		
	cb013{,a}	label		
	cb012{,a}	label		
	cba{,a}	label		

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
FBN	`fbn{,a}`	`label`	Branch on floating-point	branch never
FBU	`fbu{,a}`	`label`	condition codes	
FBG	`fbg{,a}`	`label`		
FBUG	`fbug{,a}`	`label`		
FBL	`fbl{,a}`	`label`		
FBUL	`fbul{,a}`	`label`		
FBLG	`fblg{,a}`	`label`		
FBNE	`fbne{,a}`	`label`		synonym: `fbnz`
FBE	`fbe{,a}`	`label`		synonym: `fbz`
FBUE	`fbue{,a}`	`label`		
FBGE	`fbge{,a}`	`label`		
FBUGE	`fbuge{,a}`	`label`		
FBLE	`fble{,a}`	`label`		
FBULE	`fbule{,a}`	`label`		
FBO	`fbo{,a}`	`label`		
FBA	`fba{,a}`	`label`		
FLUSH	`flush`	*address*	*Instruction cache flush*	
JMPL	`jmpl`	*address*, reg_{rd}	Jump and link	
LDSB	`ldsb`	*[address]*, reg_{rd}	Load signed byte	
LDSH	`ldsh`	*[address]*, reg_{rd}	Load signed halfword	
LDSTUB	`ldstub`	*[address]*, reg_{rd}	Load-store unsigned byte	
LDUB	`ldub`	*[address]*, reg_{rd}	Load unsigned byte	
LDUH	`lduh`	*[address]*, reg_{rd}	Load unsigned halfword	
LD	`ld`	*[address]*, reg_{rd}	Load word	
LDD	`ldd`	*[address]*, reg_{rd}	Load double word	reg_{rd} must be even
LDF	`ld`	*[address]*, $freg_{rd}$		
LDFSR	`ld`	*[address]*, %fsr	Load floating-point register	
LDDF	`ldd`	*[address]*, $freg_{rd}$	Load double floating-point Load coprocessor	$freg_{rd}$ must be even
LDC	`ld`	*[address]*, $creg_{rd}$		
LDCSR	`ld`	*[address], %csr*	Load double coprocessor	
LDDC	`ldd`	*[address]*, $creg_{rd}$		

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	**Mnemonic**	**Argument List**	**Operation**	**Comments**
`LDSBA`	`ldsba`	*[regaddr]asi*, reg_{rd}	Load signed byte from alternate space	
`LDSHA`	`ldsha`	*[regaddr]asi*, reg_{rd}		
`LDUBA`	`lduba`	*[regaddr]asi*, reg_{rd}		
`LDUHA`	`lduha`	*[regaddr]asi*, reg_{rd}		
`LDA`	`lda`	*[regaddr]asi*, reg_{rd}		
`LDDA`	`ldda`	*[regaddr]asi*, reg_{rd}		reg_{rd} must be even
`LDSTUBA`	`ldstuba`	*[regaddr]asi*, reg_{rd}		
`MULScc`	`mulscc`	reg_{rs1}, *reg_or_imm*, reg_{rd}	Multiply step (and modify `icc`)	
`NOP`	`nop`		No operation	
`OR`	`or`	reg_{rs1}, *reg_or_imm*, reg_{rd}	Inclusive `or`	
`ORcc`	`orcc`	reg_{rs1}, *reg_or_imm*, reg_{rd}		
`ORN`	`orn`	reg_{rs1}, *reg_or_imm*, reg_{rd}		
`ORNcc`	`orncc`	reg_{rs1}, *reg_or_imm*, reg_{rd}		
`RDASR`	`rd`	%asrn_{rs1}, reg_{rd}		
`RDY`	`rd`	%y, reg_{rd}		See synthetic instructions.
`RDPSR`	`rd`	%psr, reg_{rd}		See synthetic instructions.
`RDWIM`	`rd`	%wim, reg_{rd}		See synthetic instructions.
`RDTBR`	`rd`	%tbr, reg_{rd}		See synthetic instructions.
`RESTORE`	`restore`	reg_{rs1}, *reg_or_imm*, reg_{rd}		See synthetic instructions.
`RETT`	`rett`	*address*	Return from trap	
`SAVE`	`save`	reg_{rs1}, *reg_or_imm*, reg_{rd}		See synthetic instructions.
`SDIV`	`sdiv`	reg_{rs1}, *reg_or_imm*, reg_{rd}	Signed divide	
`SDIVcc`	`sdivcc`	reg_{rs1}, *reg_or_imm*, reg_{rd}	Signed divide and modify `icc`	

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
SMUL SMULcc	smul smulcc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Signed multiply Signed multiply and modify icc	
SETHI	sethi sethi	*const22*, reg_{rd} %hi(*value*), reg_{rd}	Set high 22 bits of register	 See synthetic instructions.
SLL SRL SRA	sll srl sra	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Shift left logical Shift right logical Shift right arithmetic	
STB STH ST STD STF STDF STFSR STDFQ	stb sth st std st std st std	reg_{rd}, [*address* reg_{rd}, [*address*] reg_{rd}, [*address*] reg_{rd}, [*address*] $freg_{rd}$, [*address*] $freg_{rd}$, [*address*] %fsr, [*address*] %fq, [*address*]	Store byte Store half-word Store floating-point status register Store double floating-point queue	Synonyms: stub, stsb Synonyms: stuh, stsh reg_{rd} Must be even $freg_{rd}$ Must be even
STC STDC STCSR STDCQ	st std st std	$creg_{rd}$, [*address*] $creg_{rd}$, [*address*] %csr, [*address*] %cq, [*address*]	Store coprocessor Store double coprocessor	 $creg_{rd}$ Must be even
STBA STHA STA STDA	stba stha sta stda	reg_{rd} [*regaddr*]*asi* reg_{rd} [*regaddr*]*asi* reg_{rd}, [*regaddr*]*asi* reg_{rd}, [*regaddr*]*asi*	Store byte into alternate space	Synonyms: stuba, stsba Synonyms: stuha, stsha reg_{rd} Must be even

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
SUB	sub	reg_{rs1}, *reg_or_imm*, reg_{rd}	Subtract	
SUBcc	subcc	reg_{rs1}, *reg_or_imm*, reg_{rd}	Subtract and modify `icc`	
SUBX	subx	reg_{rs1}, *reg_or_imm*, reg_{rd}	Subtract with carry	
SUBXcc	subxcc	reg_{rs1}, *reg_or_imm*, reg_{rd}		
SWAP	swap	*[address]*, reg_{rd}	Swap memory word with register	
SWAPA	swapa	*[regaddr]asi*, reg_{rd}		
Ticc	tn	software_trap_number	Trap on integer condition code	Trap never
	tne	software_trap_number	**Note:** Trap numbers 16-31 are reserved for the user. Currently-defined trap numbers are those defined in `/usr/include/sys/trap.h`	Synonym: `tnz`
	te	software_trap_number		Synonym: `tz`
	tg	software_trap_number		
	tle	software_trap_number		
	tge	software_trap_number		
	tl	software_trap_number		
	tgu	software_trap_number		
	tleu	software_trap_number		
	tlu	software_trap_number		Synonym: `tcs`
	tgeu	software_trap_number		Synonym: `tcc`
	tpos	software_trap_number		
	tneg	software_trap_number		
	tvc	software_trap_number		Synonym: `t`
	tvs	software_trap_number		
	ta	software_trap_number		
TADDcc	taddcc	reg_{rs1}, *reg_or_imm*, reg_{rd}	Tagged add and modify `icc`	
TSUBcc	tsubcc	reg_{rs1}, *reg_or_imm*, reg_{rd}		
TADDccTV	taddcctv	reg_{rs1}, *reg_or_imm*, reg_{rd}	Tagged add and modify `icc` and trap on overflow	
TSUBccTV	tsubcctv	reg_{rs1}, *reg_or_imm*, reg_{rd}		

Table 5-3 SPARC to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
UDIV UDIVcc	udiv udivcc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Unsigned divide Unsigned divide and modify icc	
UMUL	umul	reg_{rs1}, *reg_or_imm*, reg_{rd}	Unsigned multiply	
UMULcc	umulcc	reg_{rs1}, *reg_or_imm*, reg_{rd}	Unsigned multiply and modify icc	
UNIMP	unimp	const22	Illegal instruction	
WRASR	wr	*reg_or_imm*, $\%asrn_{rs1}$		
WRY	wr	reg_{rs1}, *reg_or_imm*, %y		See synthetic instructions
WRPSR	wr	reg_{rs1}, *reg_or_imm*, %psr		See synthetic instructions
WRWIM	wr	reg_{rs1}, *reg_or_imm*, %wim		See synthetic instructions
WRTBR	wr	reg_{rs1}, *reg_or_imm*, %tbr		See synthetic instructions
XNOR XNORcc	xnor xnorcc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Exclusive nor	
XOR XORcc	xor xorcc	reg_{rs1}, *reg_or_imm*, reg_{rd} reg_{rs1}, *reg_or_imm*, reg_{rd}	Exclusive or	

Floating-Point Instruction

Table 5-4 shows floating-point instructions. In cases where more than numeric type is involved, each instruction in a group is described; otherwise, only the first member of a group is described.

Table 5-4 Floating-Point Instructions

SPARC	Mnemonic*	Argument List	Description
FiTOs FiTOd FiTOq	fitos fitod fitoq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert integer to single Convert integer to double Convert integer to quad
FsTOi FdTOi FqTOi	fstoi fdtoi fqtoi	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert single to integer Convert double to integer Convert quad to integer
FsTOd FsTOq	fstod fstoq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert single to double Convert single to quad
FdTOs FdTOq	fdtos fdtoq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert double to single Convert double to quad
FqTOd FqTOs	fqtod fqtos	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert quad to double Convert quad to single
FMOVs FNEGs FABSs	fmovs fnegs fabss	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Move Negate Absolute value
FSQRTs FSQRTd FSQRTq	fsqrts fsqrtd fsqrtq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Square root
FADDs FADDd FADDq	fadds faddd faddq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	Add

* Types of Operands are denoted by the following lower-case letters:
i integer
s single
d double
q quad

Table 5-4 Floating-Point Instructions *(Continued)*

SPARC	Mnemonic*	Argument List	Description
FSUBs FSUBd FSUBq	fsubs fsubd fsubq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	Subtract
FMULs FMULd FMULq	fmuls fmuld fmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	Multiply
FdMULq FsMULd	fmulq fsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	Multiply double to quad Multiply single to double
FDIVs FDIVd FDIVq	fdivs fdivd fdivq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$ $freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	Divide
FCMPs FCMPd FCMPq	fcmps fcmpd fcmpq	$freg_{rs1}$, $freg_{rs2}$ $freg_{rs1}$, $freg_{rs2}$ $freg_{rs1}$, $freg_{rs2}$	Compare
FCMPEs FCMPEd FCMPEq	fcmpes fcmped fcmpeq	$freg_{rs1}$, $freg_{rs2}$ $freg_{rs1}$, $freg_{rs2}$ $freg_{rs1}$, $freg_{rs2}$	Compare, generate exception if not ordered

* Types of Operands are denoted by the following lower-case letters:
i integer
s single
d double
q quad

Coprocessor Instructions

All *coprocessor-operate* (cpop*n*) instructions take all operands from and return all results to coprocessor registers. The data types supported by the coprocessor are coprocessor-dependent. Operand alignment is also coprocessor-dependent. Coprocessor-operate instructions are described in Table 5-5.

If the EC (PSR_enable_coprocessor) field of the processor state register (PSR) is 0, or if a coprocessor is not present, a cpop*n* instruction causes a *cp_disabled* trap.

The conditions that cause a *cp_exception* trap are coprocessor-dependent.

Table 5-5 Coprocessor-Operate Instructions

SPARC	Mnemonic	Argument List	Name	Comments
CPop1	cpop1	*opc*, reg_{rs1}, reg_{rs2}, reg_{rd}	Coprocessor operation	
CPop2	cpop2	*opc*, reg_{rs1}, reg_{rs2}, reg_{rd}	Coprocessor operation	May modify ccc

Synthetic Instructions

Table 5-6 describes the mapping of synthetic instructions to hardware instructions.

Table 5-6 Synthetic Instruction to Hardware Instruction Mapping

Synthetic Instruction		Hardware Equivalent(s)		Comment
btst bset bclr btog	*reg_or_imm*, reg_{rs1} *reg_or_imm*, reg_{rd} *reg_or_imm*, reg_{rd} *reg_or_imm*, reg_{rd}	andcc or andn xor	reg_{rs1}, *reg_or_imm*, %g0 reg_{rd}, *reg_or_imm*, reg_{rd} reg_{rd}, *reg_or_imm*, reg_{rd} reg_{rd}, *reg_or_imm*, reg_{rd}	Bit test Bit set Bit clear Bit toggle
call	reg_or_imm	jmpl	*reg_or_imm*, %o7	
clr clrb clrh clr	reg_{rd} [*address*] [*address*] [*address*]	or stb sth st	%g0, %g0, reg_{rd} %g0, [*address*] %g0, [*address*] %g0, [*address*]	Clear (zero) register Clear byte Clear halfword Clear word
cmp	reg, reg_or_imm	subcc	reg_{rs1}, *reg_or_imm*, %g0	Compare
dec dec deccc deccc	reg_{rd} *const13*, reg_{rd} reg_{rd} *const13*, reg_{rd}	sub sub subcc subcc	reg_{rd}, *1*, reg_{rd} reg_{rd}, *const13*, reg_{rd} reg_{rd}, *1*, reg_{rd} reg_{rd}, *const13*, reg_{rd}	Decrement by 1 Decrement by *const13* Decrement by 1 and set icc Decrement by *const13* and set icc

Table 5-6 Synthetic Instruction to Hardware Instruction Mapping *(Continued)*

Synthetic Instruction		**Hardware Equivalent(s)**		**Comment**

`inc`
`inc`
`inccc`
 	
`inccc`
	reg_{rd}
	const13, reg_{rd}
	reg_{rd}
	
	const13, reg_{rd}
		`add`
		`add`
		`addcc`
		
		`addcc`
			reg_{rd}, *1*, reg_{rd}
			reg_{rd}, *const13*, reg_{rd}
			reg_{rd}, *1*, reg_{rd}
			
			reg_{rd}, *const13*, reg_{rd}
				Increment by 1
				Increment by *const13*
				Increment by 1
				and set `icc`
				Increment by
				const13 and
				set `icc`

`jmp`	*address*	`jmpl`	*address*, %g0	

`mov`
`mov`
`mov`
`mov`
`mov`
`mov`
`mov`
`mov`
`mov`
	reg_or_imm,reg_{rd}
	%y, reg_{rs1}
	%psr, reg_{rs1}
	%wim, reg_{rs1}
	%tbr, reg_{rs1}
	reg_or_imm, %y
	reg_or_imm, %psr
	reg_or_imm, %wim
	reg_or_imm, %tbr
		`or`
		`rd`
		`rd`
		`rd`
		`rd`
		`wr`
		`wr`
		`wr`
		`wr`
			%g0, *reg_or_imm*, reg_{rd}
			%y, reg_{rs1}
			%psr, reg_{rs1}
			%wim, reg_{rs1}
			%tbr, reg_{rs1}
			%g0,*reg_or_imm*,%y
			%g0,*reg_or_imm*,%psr
			%g0,*reg_or_imm*,%wim
			%g0,*reg_or_imm*,%tbr	

`not`
`not`
`neg`
`neg`
	reg_{rs1}, reg_{rd}
	reg_{rd}
	reg_{rs1}, reg_{rd}
	reg_{rd}
		`xnor`
		`xnor`
		`sub`
		`sub`
			reg_{rs1}, %g0, reg_{rd}
			reg_{rd}, %g0, reg_{rd}
			%g0, reg_{rs2}, reg_{rd}
			%g0, reg_{rd}, reg_{rd}
				One's complement
				One's complement
				Two's complement
				Two's complement

`restore`		`restore`	%g0, %g0, %g0	Trivial *restore*
`save`		`save`	%g0, %g0, %g0	Trivial *save* *trivial save* should only be used in supervisor code!

`set`
`set`
`set`
	value,reg_{rd}
	value,reg_{rd}
	value,reg_{rd}
		`or`
		`sethi`
		`sethi`
		`or`
			%g0, *value*, reg_{rd}
			%hi(*value*), reg_{rd}
			%hi(*value*), reg_{rd};
			reg_{rd}, %lo(*value*), reg_{rd}
				if -4096 ≤ *value* ≤ 4095
				if ((*value* & 0x3ff) == 0)
				otherwise
				Do not
				use the `set` synthetic
				instruction in an
				instruction delay
				slot.

Table 5-6 Synthetic Instruction to Hardware Instruction Mapping *(Continued)*

Synthetic Instruction		Hardware Equivalent(s)		Comment
`skipz` `skipnz`		`bnz,a .+8` `bz,a .+8`		if *z* is set, ignores next instruction if *z* is not set, ignores next instruction
`tst`	*reg*	`orcc`	reg_{rs1}, %g0, %g0	test