SPARC-V9 Instruction Set E

This appendix describes changes made to the SPARC instruction set due to the SPARC-V9 architecture. Application software for the 32-bit SPARC-V8 (Version8) architecture can execute, unchanged, on SPARC-V9 systems.

This appendix is organized into the following sections:

SPARC-V9 Changes

The SPARC-V9 architecture differs from SPARC-V8 architecture in the following areas, expanded below: registers, alternate space access, byte order, and instruction set.

Registers

These registers have been deleted:

Table E-1 Deleted SPARC-V8 Privileged Registers

PSR	Processor State Register
TBR	Trap Base Register
WIM	Window Invalid Mask

These registers have been widened from 32 to 64 bits:

Table E-2 Registers Widened from 32 to 64 bits

Integer registers	
All state registers	FSR, PC, nPC, and Y

Note – FSR Floating-Point State Register: *fcc1*, *fcc2*, and *fcc3* (added floating-point condition code) bits are added and the register widened to 64-bits.

These SPARC-V9 registers are within a SPARC-V8 register field:

Table E-3 SPARC-V9 Registers Within a SPARC-V8 Register Field

CCR	Condition Codes Register
CWP	Current Window Pointer
PIL	Processor Interrupt Level
TBA	Trap Base Address
TT[MAXTL]	Trap Type
VER	Version

These are registers that have been added.

Table E-4 Registers That have Been Added

ASI	Address Space Identifier
CANRESTORE	Restorable Windows
CANSAVE	Savable windows

Table E-4 Registers That have Been Added *(Continued)*

CLEANWIN	Clean Windows
FPRS	Floating-point Register State
OTHERWIN	Other Windows
PSTATE	Processor State
TICK	Hardware clock tick-counter
TL	Trap Level
TNPC[MAXTL]	Trap Next Program Counter
TPC[MAXTL]	Trap Program Counter
TSTATE[MAXTL]	Trap State
WSTATE	Windows State

Also, there are sixteen additional double-precision floating-point registers, *f*[32] .. *f*[62]. These registers overlap (and are aliased with) eight additional quad-precision floating-point registers, f[32] .. f[60]

The SPARC-V9, CWP register is decremented during a RESTORE instruction, and incremented during a SAVE instruction. This is the opposite of PSR.CWP's behavior in SPARC-V8. This change has no effect on nonprivileged instructions.

Alternate Space Access

Load- and store-alternate instructions to one-half of the alternate spaces can now be included in user code. In SPARC-V9, loads and stores to ASIs 00_{16} .. $7f_{16}$ are privileged; those to ASIs 80_{16} .. FF_{16} are nonprivileged. In SPARC-V8, access to alternate address spaces is privileged.

Byte Order

SPARC-V9 supports both little- and big-endian byte orders for data accesses only; instruction accesses are always performed using big-endian byte order. In SPARC-V8, all data and instruction accesses are performed in big-endian byte order.

SPARC-V9 Instruction Set Changes

Application software written for the SPARC-V8 processor runs unchanged on a SPARC-V9 processor.

Extended Instruction Definitions to Support the 64-bit Model

Table E-5 Extended Instruction Definitions for 64-bit Model

FCMP, FCMPE	Floating-Point Compare—can set any of the four floating-point condition codes.
LDFSR, STFSR	Load/Store FSR- only affect low-order 32 bits of FSR
LDUW, LDUWA	Same as LD, LDA in SPARC-V8
RDASR/WRASR	Read/Write State Registers - access additional registers
SAVE/RESTORE	
SETHI	
SRA, SRL, SLL, Shifts	Split into 32-bit and 64-bit versions
Tcc	(was Ticc) Operates with either the 32-bit integer condition codes (*icc*), or the 64-bit integer condition codes (*xcc*)

All other arithmetic operations operate on 64-bit operands and produce 64-bit results.

Added Instructions to Support 64 bits

Table E-6 Added Instructions to Support 64 bits

F[sdq]TOx	Convert floating point to 64-bit word
FxTO[sdq]	Convert 64-bit word to floating point
FMOV[dq]	Floating-Point Move, double and quad
FNEG[dq]	Floating-point Negate, double and quad
FABS[dq]	Floating-point Absolute Value, double and quad
LDDFA, STDFA, LDFA, STFA	Alternate address space forms of LDDF, STDF, LDF, and STF
LDSW	Load a signed word
LDSWA	Load a signed word from an alternate space
LDX	Load an extended word
LDXA	Load an extended word from an alternate space
LDXFSR	Load all 64 bits of the FSR register
STX	Store an extended word
STXA	Store an extended word into an alternate space
STXFSR	Store all 64 bits if the FSR register

Added Instructions to Support High-Performance System Implementation

Table E-7 Added Instructions to Support High-Performance

BPcc	Branch on integer condition code with prediction
BPr	Branch on integer register contents with prediction
CASA, CASXA	Compare and Swap from an alternate space
FBPfcc	Branch on floating-point condition code with prediction
FLUSHW	Flush windows
FMOVcc	Move floating-point register if condition code is satisfied
FMOVr	Move floating-point register if integer register satisfies condition
LDQF(A), STQF(A)	Load/Store Quad Floating-point (in an alternate space)

Table E-7 Added Instructions to Support High-Performance (Continued)

MOVcc	Move integer register if condition code is satisfied
MOVr	Move integer register if register contents satisfy condition
MULX	Generic 64-bit multiply
POPC	Population count
PREFETCH, PREFETCHA	Prefetch Data
SDIVX, UDIVX	Signed and Unsigned 64-bit divide

Deleted Instructions

Table E-8 Deleted Instructions

Coprocessor loads and stores	
RDTBR and WRTBR	TBR no longer exists. It is replaced by TBA, which can be read/written with RDPR/WRPR instructions
RDWIM and WRWIM	WIM no longer exists. WIM has been replaced by several register-window registers
REPSR and WRPSR	PSR no longer exists. It has been replaced by several separate registers that are read/written with other instructions
RETT	Return from trap (replace by DONE/RETRY)
STDFQ	Store Double from Floating-point Queue (replaced by the RDPR FQ instruction

Miscellaneous Instruction Changes

Table E-9 Miscellaneous Instruction Changes

IMPDEPn	(Changed) Implementation-dependent instructions (replace SPARC-V8 CPop instructions)
MEMBAR	(Added) Memory barrier (memory synchronization support)

SPARC-V9 Instruction Set Mapping

The following tables describe the SPARC-V9 instruction-set mapping.

Table E-10 SPARC-V9 to Assembly Language Mapping

Opcode	Mnemonic	Argument List	Operation	Comments
			(Branch on cc with prediction)	
BPA	ba{,a} {,pt|,pn}	%icc or %xcc, label	Branch always	1
BPN	bn{,a} {,pt|,pn}	%icc or %xcc, label	Branch never	0
BPNE	bne{,a} {,pt|,pn}	%icc or %xcc, label	Branch on not equal	not Z
BPE	be{,a} {,pt|,pn}	%icc or %xcc, label	Branch on equal	Z
BPG	bg{,a} {,pt|,pn}	%icc or %xcc, label	Branch on greater	not (Z or (N xor V))
BPLE	ble{,a} {,pt|,pn}	%icc or %xcc, label	Branch on less or equal	Z or (N xor V)
BPGE	bge{,a} {,pt|,pn}	%icc or %xcc, label	Branch on greater or equal	not (N xor V)
BPL	bl{,a} {,pt|,pn}	%icc or %xcc, label	Branch on less	N xor V
BPGU	bgu{,a} {,pt|,pn}	%icc or %xcc, label	Branch on greater unsigned	not (C or Z)
BPLEU	bleu{,a} {,pt|,pn}	%icc or %xcc, label	Branch on less or equal unsigned	C or Z
BPCC	bcc{,a} {,pt|,pn}	%icc or %xcc, label	Branch on carry clear (greater than or equal, unsigned)	not C
BPCS	bcs{,a} {,pt|,pn}	%icc or %xcc, label	Branch on carry set (less than, unsigned)	C
BPPOS	bpos{,a} {,pt|,pn}	%icc or %xcc, label	Branch on positive	not N
BPNEG	bneg{,a} {,pt|,pn}	%icc or %xcc, label	Branch on negative	N
BPVC	bvc{,a} {,pt|,pn}	%icc or %xcc, label	Branch on overflow clear	not V
BPVS	bvs{,a} {,pt|,pn}	%icc or %xcc, label	Branch on overflow set	V

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
BRZ	brz{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register zero	Z
BRLEZ	brlez{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register less than or equal to zero	N or Z
BRLZ	brlz{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register less than zero	N
BRNZ	brnz{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register not zero	not Z
BRGZ	brgz{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register greater than zero	not (N or Z)
BRGEZ	brgez{,a} {,pt|,pn}	reg_{rs1}, label	Branch on register greater than or equal to zero	not N
CASA	casa casa	[reg_{rs1}]*imm_asi*,reg_{rs2},reg_{rd} [reg_{rs1}]%asi,reg_{rs2},reg_{rd}	Compare and swap word from alternate space	
CASXA	casxa casxa	[reg_{rs1}]*imm_asi*,reg_{rs2},reg_{rd} [reg_{rs1}]%asi,reg_{rs2},reg_{rd}	Compare and swap extended from alternate space	

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
			(Branch on cc with prediction)	
FBPA	fba{,a} {,pt|,pn}	%fcc*n*, label	Branch always	1
FBPN	fbn{,a} {,pt|,pn}	%fcc*n*, label	Branch never	0
FBPU	fbu{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered	U
FBPG	fbg{,a} {,pt|,pn}	%fcc*n*, label	Branch on greater	G
FBPUG	fbug{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered or greater	G or U
FBPL	fbl{,a} {,pt|,pn}	%fcc*n*, label	Branch on less	L
FBPUL	fbul{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered or less	L or U
FBPLG	fblg{,a} {,pt|,pn}	%fcc*n*, label	Branch on less or greater	L or G
FBPNE	fbne{,a} {,pt|,pn}	%fcc*n*, label	Branch on not equal	L or G or U
FBPE	fbe{,a} {,pt|,pn}	%fcc*n*, label	Branch on equal	E
FBPUE	fbue{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered or equal	E or U
FBPGE	fbge{,a} {,pt|,pn}	%fcc*n*, label	Branch on greater or equal	E or G
FBPUGE	fbuge{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered or greater or equal	E or G or U
FBPLE	fble{,a} {,pt|,pn}	%fcc*n*, label	Branch on less or equal	E or L
FBPULE	fbule{,a} {,pt|,pn}	%fcc*n*, label	Branch on unordered or less or equal	E or L or u
FBPO	fbo{,a} {,pt|,pn}	%fcc*n*, label	Branch on ordered	E or L or G
FLUSHW	flushw		Flush register windows	

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
			(Move on integer cc)	
FMOVA	fmov{s,d,q}a	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move always	1
FMOVN	fmov{s,d,q}n	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move never	0
FMOVNE	fmov{s,d,q}ne	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if not equal	not Z
FMOVE	fmov{s,d,q}e	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if equal	Z
FMOVG	fmov{s,d,q}g	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if greater	not (Z or (N xor V))
FMOVLE	fmov{s,d,q}le	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if less or equal	Z or (N xor V)
FMOVGE	fmov{s,d,q}ge	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if greater or equal	not (N xor V)
FMOVL	fmov{s,d,q}l	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if less	N xor V
FMOVGU	fmov{s,d,q}gu	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if greater unsigned	not (C or Z)
FMOVLEU	fmov{s,d,q}leu	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if less or equal unsigned	C or Z
FMOVCC	fmov{s,d,q}cc	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if carry clear (greater or equal, unsigned)	not C
FMOVCS	fmov{s,d,q}cs	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if carry set (less than, unsigned)	C
FMOVPOS	fmov{s,d,q}pos	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if positive	not N
FMOVNEG	fmov{s,d,q}neg	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if negative	N
FMOVVC	fmov{s,d,q}vc	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if overflow clear	not V
FMOVVS	fmov{s,d,q}vs	%icc or %xcc, $freg_{rs2}$, $freg_{rd}$	Move if overflow set	V

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
			(Move f-p register on cc)	
FMOVRZ	fmovr {s,d,q}e	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register zero	
FMOVRLEZ	fmovr {s,d,q}lz	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register less than or equal zero	
FMOVRLZ	fmovr {s,d,q}lz	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register less than zero	
FMOVRNZ	fmovr {s,d,q}ne	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register not zero	
FMOVRGZ	fmovr {s,d,q}gz	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register greater than zero	
FMOVRGEZ	fmovr {s,d,q}gez	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$	Move if register greater than or equal to zero	

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
			(Move on floating-point cc)	
FMOVFA	fmov{s,d,q}a	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move always	1
FMOVFN	fmov{s,d,q}n	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move never	0
FMOVFU	fmov{s,d,q}u	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered	U
FMOVFG	fmov{s,d,q}g	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if greater	G
FMOVFUG	fmov{s,d,q}ug	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered or greater	G or U
FMOVFL	fmov{s,d,q}l	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if less	L
FMOVFUL	fmov{s,d,q}ul	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered or less	L or U
FMOVFLG	fmov{s,d,q}lg	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if less or greater	L or G
FMOVFNE	fmov{s,d,q}ne	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if not equal	L or G or U
FMOVFE	fmov{s,d,q}e	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if equal	E
FMOVFUE	fmov{s,d,q}ue	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered or equal	E or U
FMOVFGE	fmov{s,d,q}ge	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if greater or equal	E or G
FMOVFUGE	fmov{s,d,q}uge	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered or greater or equal	E or G or U
FMOVFLE	fmov{s,d,q}le	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if less or equal	E or L
FMOVFULE	fmov{s,d,q}ule	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if unordered or less or equal	E or L or u
FMOVFO	fmov{s,d,q}o	%fcc*n*, $freg_{rs2}$, $freg_{rd}$	Move if ordered	E or L or G
LDSW LDSWA	ldsw ldsw	*[address]*, reg_{rd} *[regaddr] imm_asi*, reg_{rd}	Load a signed word Load signed word from alternate space	

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
LDX LDXA LDXFSR	ldx ldxa ldxa ldx	*[address]*, reg_{rd} *[regaddr] imm_asi*, reg_{rd} *[reg_plus_imm]* %asi, reg_{rd} *[address]*, %fsr	Load extended word Load extended word from alternate space Load floating-point state register	
MEMBAR	membar	*membar_mask*	Memory barrier	
			(Move integer register on cc)	
MOVA	mova	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move always	1
MOVN	movn	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move never	0
MOVNE	movne	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if not equal	not Z
MOVE	move	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if equal	Z
MOVG	movg	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if greater	not (Z or (N xor V))
MOVLE	movle	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if less or equal	Z or (N xor V)
MOVGE	movge	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if greater or equal	not (N xor V)
MOVL	movl	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if less	N xor V
MOVGU	movgu	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if greater unsigned	not (C or Z)
MOVLEU	movleu	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if less or equal unsigned	C or Z
MOVCC	movcc	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if carry clear (greater or equal, unsigned)	not C
MOVCS	movcs	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if carry set (less than, unsigned)	C
MOVPOS	movpos	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if positive	not N
MOVNEG	movneg	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if negative	N
MOVVC	movvc	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if overflow clear	not V
MOVVS	movvs	%icc or %xcc, *reg_or_imm11*, reg_{rd}	Move if overflow set	V

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
			(Move on floating-point cc)	
MOVFA	mova	%fccn, *reg_or_imm11*, reg_{rd}	Move always	1
MOVFN	movn	%fccn, *reg_or_imm11*, reg_{rd}	Move never	0
MOVFU	movu	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered	U
MOVFG	movg	%fccn, *reg_or_imm11*, reg_{rd}	Move if greater	G
MOVFUG	movug	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered or greater	G or U
MOVFL	movl	%fccn, *reg_or_imm11*, reg_{rd}	Move if less	L
MOVFUL	movul	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered or less	L or U
MOVFLG	movlg	%fccn, *reg_or_imm11*, reg_{rd}	Move if less or greater	L or G
MOVFNE	movne	%fccn, *reg_or_imm11*, reg_{rd}	Move if not equal	L or G or U
MOVFE	move	%fccn, *reg_or_imm11*, reg_{rd}	Move if equal	E
MOVFUE	movue	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered or equal	E or U
MOVFGE	movge	%fccn, *reg_or_imm11*, reg_{rd}	Move if greater or equal	E or G
MOVFUGE	movuge	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered or greater or equal	E or G or U
MOVFLE	movle	%fccn, *reg_or_imm11*, reg_{rd}	Move if less or equal	E or L
MOVFULE	movule	%fccn, *reg_or_imm11*, reg_{rd}	Move if unordered or less or equal	E or L or u
MOVFO	movo	%fccn, *reg_or_imm11*, reg_{rd}	Move if ordered	E or L or G
			(Move register on register cc)	
MOVRZ	movre	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register zero	Z
MOVRLEZ	movrlez	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register less than or equal to zero	N or Z
MOVRLZ	movrlz	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register less than zero	N
MOVRNZ	movrnz	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register not zero	not Z
MOVRGZ	movrgz	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register greater than zero	N nor Z
MOVRGEZ	movrgez	reg_{rs1}, *reg_or_imm10*,reg_{rd}	Move if register greater than or equal to zero	not N
			(Generic 64-bit Multiply)	See SDIVX
MULX	mulx	reg_{rs1}, *reg_or_imm*,reg_{rd}	Multiply (signed or unsigned)	and UDIVX
POPC	popc	*reg_or_imm*, reg_{rd}	Population count	
PREFETCH	prefetch	*[address], prefetch_dcn*	Prefetch data	See *The SPARC architecture manual*, version 9
PREFETCHA	prefetcha	*[regaddr] imm_asi, prefetch_fcn*	Prefetch data from alternate space	
	prefetcha	*[reg_plus_imm]* %asi, *prefetch_fcn*		

Table E-10 SPARC-V9 to Assembly Language Mapping *(Continued)*

Opcode	Mnemonic	Argument List	Operation	Comments
SDIVX	sdivx	reg_{rs1}, *reg_or_imm*,reg_{rd}	(64-bit signed divide) Signed Divide	See MULX and UDIVX
STX STXA STXFSR	stx stxa stxa stx	reg_{rd}, *[address]* reg_{rd}, *[address] imm_asi* reg_{rd}, *[reg_plus_imm]* %asi %fsr, *[address]*	Store extended word Store extended word into alternate space Store floating-point register (all 64-bits)	
UDIVX	udivx	reg_{rs1}, *reg_or_imm*, reg_{rd}	(64-bit unsigned divide) Unsigned divide	See MULX and SDIVX

SPARC-V9 Floating-Point Instruction Set Mapping

SPARC-V9 floating-point instructions are shown in Table E-11

Table E-11 SPARC-V9 Floating-Point Instructions

SPARC	Mnemonic*	Argument List	Description
F[sdq]TOx	fstox fdtox fqtox fstoi fdtoi fqtoi	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert floating point to 64-bit integer Convert floating-point to 32-bit integer

* Types of Operands are denoted by the following lower-case letters:
i 32-bit integer
x 64-bit integer
s single
d double
q quad

Table E-11 SPARC-V9 Floating-Point Instructions (Continued)

SPARC	Mnemonic*	Argument List	Description
FxTO[sdq]	fxtos fxtod fxtoq fitos fitod fitoq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Convert 64-bit integer to floating point Convert 32-bit integer to floating point
FMOV[dq]	fmovd fmovq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Move double Move quad
FNEG[dq]	fnegd fnegq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Negate double Negate quad
FABS[dq]	fabsd fabsq	$freg_{rs2}$, $freg_{rd}$ $freg_{rs2}$, $freg_{rd}$	Absolute value double Absolute value quad
LDFA LDDFA LDQFA	lda lda ldda ldda ldqa ldqa	[*regaddr*] *imm_asi*, $freg_{rd}$ [*reg_plus_imm*] %asi, $freg_{rd}$ [*regaddr*] *imm_asi*, $freg_{rd}$ [*reg_plus_imm*] %asi, $freg_{rd}$ [*regaddr*] *imm_asi*, $freg_{rd}$ [*reg_plus_imm*] %asi, $freg_{rd}$	Load floating-point register from alternate space Load double floating-point register from alternate space. Load quad floating-point register from alternate space
STFA STDFA STQFA	sta sta stda stda stqa stqa	$freg_{rd}$, [*regaddr*] *imm_asi* $freg_{rd}$, [*reg_plus_imm*] %asi $freg_{rd}$, [*regaddr*] *imm_asi* $freg_{rd}$, [*reg_plus_imm*] %asi $freg_{rd}$, [*regaddr*] *imm_asi* $freg_{rd}$, [*reg_plus_imm*] %asi	Store floating-point register to alternate space Store double floating-point register to alternate space Store quad floating-point register to alternate space

* Types of Operands are denoted by the following lower-case letters:
`i` 32-bit integer
`x` 64-bit integer
`s` single
`d` double
`q` quad

SPARC-V9 Synthetic Instruction-Set Mapping

Here is a mapping of synthetic instructions to hardware equivalent instructions.

Table E-12 SPARC-V9 Synthetic Instructions to Hardware Instruction

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

`cas`
`casl`
`casx`
`casxl`
	[reg_{rs1}], reg_{rs2}, reg_{rd}
	[reg_{rs1}], reg_{rs2}, reg_{rd}
	[reg_{rs1}], reg_{rs2}, reg_{rd}
	[reg_{rs1}], reg_{rs2}, reg_{rd}
		`casa`
		`casa`
		`casxa`
		`casxa`
			[reg_{rs1}]ASI_P, reg_{rs2}, reg_{rd}
			[reg_{rs1}]ASI_P_L, reg_{rs2}, reg_{rd}
			[reg_{rs1}]ASI_P, reg_{rs2}, reg_{rd}
			[reg_{rs1}]ASI_P_L, reg_{rs2}, reg_{rd}
				Compare & swap (cas)
				cas little-endian
				cas extended
				cas little-endian, extended

`clrx`	[*address*]	`stx`	%g0, [*address*]	Clear extended word
`clruw` `clruw`	reg_{rs1}, reg_{rd} reg_{rd}	`srl` `srl`	reg_{rs1}, %g0, reg_{rd} reg_{rd}, %g0, reg_{rd}	Copy and clear upper word Clear upper word
`iprefetch`	*label*	`bn, pt`	%xcc, *label*	Instruction prefetch,

`mov`
`mov`
`mov`
	%y, reg_{rd}
	%asr*n*, reg_{rd}
	reg_or_imm, %asr*n*
		`rd`
		`rd`
		`wr`
			%y, reg_{rd}
			%asr*n*, reg_{rd}
			%g0, reg_or_imm, %asr*n*	

`ret` `retl`		`jmpl` `jmpl`	%i7+8, %g0 %o7+8, %g0	Return from subroutine Return from *leaf* subroutine

`setuw`	*value*,reg_{rd}	`sethi`
		`or`
		`sethi`
		`or`
			%hi(*value*), reg_{rd}
			%g0, *value*, reg_{rd}
			%hi(*value*), reg_{rd};
			reg_{rd}, %lo(*value*), reg_{rd}
				(*value* & $3FF_{16}$)==0
				when 0 ≤ *value* ≤ 4095
				(otherwise)
				Do not use `setuw` in a DCTI delay slot.

Table E-12 SPARC-V9 Synthetic Instructions to Hardware Instruction *(Continued)*

Synthetic Instruction		Hardware Equivalent(s)		Comment
`setsw`	*value*,reg_{rd}	`sethi`	%hi(*value*), reg_{rd}	*value*>=0 and (*value* & $3FF_{16}$)==0
		`or`	%g0, *value*, reg_{rd}	-4096 ≤ *value* ≤ 4095
		`sethi`	%hi(*value*), reg_{rd}	if (*value*<0) and ((*value* & 3FF)==0)
		`sra`	reg_{rd}, %g0, reg_{rd}	
		`sethi`	%hi(*value*), reg_{rd};	(otherwise, if *value*>=0)
		`or`	reg_{rd}, %lo(*value*), reg_{rd}	
		`sethi`	%hi(*value*), reg_{rd};	(otherwise, if *value*<0)
		`or`	reg_{rd}, %lo(*value*), reg_{rd}	
		`sra`	reg_{rd}, %g0, reg_{rd}	Do not use `setsw` in a CTI delay slot.
`signx`	reg_{rs1}, reg_{rd}	`sra`	reg_{rs1}, %g0, reg_{rd}	Sign-extend 32-bit value to 64 bits
`signx`	reg_{rd}	`sra`	reg_{rd}, %g0, reg_{rd}	

SPARC-V9 Instruction Set Extensions

This section describes extensions to the SPARC-V9 instruction set. The extensions support enhanced graphics functionality and improved memory access efficiency.

Note – SPARC-V9 instruction set extensions used in executables may not be portable to other SPARC-V9 systems.

Graphics Data Formats

The overhead of converting to and from floating-point arithmetic is high, so the graphics instructions are optimized for short-integer arithmetic. Image components are 8 or 16 bits. Intermediate results are 16 or 32 bits.

Eight-bit Format

A 32-bit word contains pixels of four unsigned 8-bit integers. The integers represent image intensity values (α, G, B, R). Support is provided for *band interleaved* images (store color components of a point), and *band sequential* images (store all values of one color component).

Fixed Data Formats

A 64-bit word contains four 16-bit signed fixed-point values. This is the fixed 16-bit data format.

A 64-bit word contains two 8-bit signed fixed-point values. This is the fixed 32-bit data format.

Enough precision and dynamic range (for filtering and simple image computations on pixel values) can be provided by an intermediate format of fixed data values. Pixel multiplication is used to convert from pixel data to fixed data. Pack instructions are used to convert from fixed data to pixel data (clip and truncate to an 8-bit unsigned value). The FPACKFIX instruction supports conversion from 32-bit fixed to 16-bit fixed. Rounding is done by adding one to the rounding bit position. You should use floating-point data to perform complex calculations needing more precision or dynamic range.

SHUTDOWN Instruction

All outstanding transactions are completed before the SHUTDOWN instruction completes.

Table E-13 SPARC-V9 SHUTDOWN Instruction

SPARC	Mnemonic	Argument List	Description
`SHUTDOWN`	`shutdown`		shutdown to enter power down mode

Graphics Status Register (GSR)

You use ASR 0x13 instructions RDASR and WRASR to access the Graphics Status Register.

Table E-14 Graphics Status Register

SPARC	Mnemonic	Argument List	Description
`RDASR` `WRASR`	`rdasr` `wrasr`	%gsr, reg_{rd} reg_{rs1}, *reg_or_imm*, %gsr	read GSR write GSR

Graphics Instructions

Unless otherwise specified, floating-point registers contain all instruction operands. There are 32 double-precision registers. Single-precision floating-point registers contain the pixel values, and double-precision floating-point registers contain the fixed values.

The opcode space reserved for the Implementation-Dependent Instruction1 (IMPDEP1) instructions is where the graphics instruction set is mapped.

Partitioned add/subtract instructions perform two 32-bit or four 16-bit partitioned adds or subtracts between the source operands corresponding fixed point values.

Table E-15 SPARC-V9 Partitioned Add/Subtract

SPARC	Mnemonic	Argument List	Description
FPADD16	fpadd16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	four 16-bit add
FPADD16S	fpadd16s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 16-bit add
FPADD32	fpadd32	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 32-bit add
FPADD32S	fpadd32s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	one 32-bit add
FPSUB16	fpsub16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	four 16-bit subtract
FPSUB16S	fpsub16s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 16-bit subtract
FPSUB32	fpsub32	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 32-bit subtract
FPSUB32S	fpsub32s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	one 32-bit subtract

Pack instructions convert to a lower pixel or precision fixed format.

Table E-16 SPARC-V9 Pixel Formatting

SPARC	Mnemonic	Argument List	Description
FPACK16	fpack16	$freg_{rs2}$, $freg_{rd}$	four 16-bit packs
FPACK32	fpack32	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 32-bit packs
FPACKFIX	fpackfix	$freg_{rs2}$, $freg_{rd}$	four 16-bit packs
FEXPAND	fexpand	$freg_{rs2}$, $freg_{rd}$	four 16-bit expands
FPMERGE	fpmerge	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	two 32-bit merges

Partitioned multiply instructions have the following variations.

Table E-17 SPARC-V9 Partitioned Multiply

SPARC	Mnemonic	Argument List	Description
FMUL8x16	fmul8x16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	8x16-bit partition
FMUL8x16AU	fmul8x16au	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	8x16-bit upper α partition
FMUL8x16AL	fmul8x16al	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	8x16-bit lower α partition
FMUL8SUx16	fmul8sux16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	upper 8x16-bit partition
FMUL8ULx16	fmul8ulx16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	lower unsigned 8x16-bit partition
FMULD8SUx16	fmuld8sux16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	upper 8x16-bit partition
FMULD8ULx16	fmuld8ulx16	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	lower unsigned 8x16-bit partition

Alignment instructions have the following variations.

Table E-18 SPARC-V9 Alignment Instructions

SPARC	Mnemonic	Argument List	Description
ALIGNADDRESS	alignaddr	reg_{rs1}, reg_{rs2}, reg_{rd}	find misaligned data access address
ALIGNADDRESS_ LITTLE	alignaddrl	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, but little-endian
FALIGNDATA	faligndata	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	do misaligned data, data alignment

Logical operate instructions perform one of sixteen 64-bit logical operations between *rs1* and *rs2* (in the standard 64-bit version).

Table E-19 SPARC-V9 Logical Operate Instructions

SPARC	Mnemonic	Argument List	Description
FZERO	fzero	$freg_{rd}$	zero fill
FZEROS	fzeros	$freg_{rd}$	zero fill, single precision
FONE	fone	$freg_{rd}$	one fill
FONES	fones	$freg_{rd}$	one fill, single precision
FSRC1	fsrc1	$freg_{rs1}$, $freg_{rd}$	copy src1
FSRC1S	fsrc1s	$freg_{rs1}$, $freg_{rd}$	copy src1, single precision
FSRC2	fsrc2	$freg_{rs2}$, $freg_{rd}$	copy src2
FSRC2S	fsrc2s	$freg_{rs2}$, $freg_{rd}$	copy src2, single precision
FNOT1	fnot1	$freg_{rs1}$, $freg_{rd}$	negate src1, 1's complement
FNOT1S	fnot1s	$freg_{rs1}$, $freg_{rd}$	same as above, single precision
FNOT2	fnot2	$freg_{rs2}$, $freg_{rd}$	negate src2, 1's complement
FNOT2S	fnot2s	$freg_{rs2}$, $freg_{rd}$	same as above, single precision
FOR	for	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical OR
FORS	fors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical OR, single precision
FNOR	fnor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical NOR
FNORS	fnors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical NOR, single precision
FAND	fand	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical AND
FANDS	fands	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical AND, single precision
FNAND	fnand	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical NAND
FNANDS	fnands	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical NAND, single precision
FXOR	fxor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical XOR
FXORS	fxors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical XOR, single precision
FXNOR	fxnor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical XNOR
FXNORS	fxnors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	logical XNOR, single precision
FORNOT1	fornot1	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	negated src1 OR src2
FORNOT1S	fornot1s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	same as above, single precision
FORNOT2	fornot2	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	src1 OR negated src2
FORNOT2S	fornot2s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	same as above, single precision
FANDNOT1	fandnot1	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	negated src1 AND src2
FANDNOT1S	fandnot1s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	same as above, single precision
FANDNOT2	fandnot2	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	src1 AND negated src2
FANDNOT2S	fandnot2s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	same as above, single precision

Pixel compare instructions compare fixed-point values in *rs1* and *rs2* (two 32 bit or four 16 bit)

Table E-20 SPARC-V9 Pixel Compare

SPARC	Mnemonic	Argument List	Description
FCMPGT16	fcmpgt16	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	4 16-bit compare, set rd if src1>src2
FCMPGT32	fcmpgt32	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	2 32-bit compare, set rd if src1>src2
FCMPLE16	fcmple16	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	4 16-bit compare, set rd if src1≤src2
FCMPLE32	fcmple32	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	2 32-bit compare, set rd if src1≤src2
FCMPNE16	fcmpne16	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	4 16-bit compare, set rd if src1≠src2
FCMPNE32	fcmpne32	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	2 32-bit compare, set rd if src1≠src2
FCMPEQ16	fcmpeq16	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	4 16-bit compare, set rd if src1=src2
FCMPEQ32	fcmpeq32	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}	2 32-bit compare, set rd if src1=src2

Edge handling instructions handle the boundary conditions for parallel pixel scan line loops.

Table E-21 SPARC V-9 Edge Handling

SPARC	Mnemonic	Argument List	Description
EDGE8	edge8	reg_{rs1}, reg_{rs2}, reg_{rd}	8 8-bit edge boundary processing
EDGE8L	edge8l	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, little-endian
EDGE16	edge16	reg_{rs1}, reg_{rs2}, reg_{rd}	4 16-bit edge boundary processing
EDGE16L	edge16l	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, little-endian
EDGE32	edge32	reg_{rs1}, reg_{rs2}, reg_{rd}	2 32-bit edge boundary processing
EDGE32L	edge32l	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, little-endian

Pixel component distance instructions are used for motion estimation in video compression algorithms.

SPARC-V9 Pixel Component Distance Instruction

SPARC	Mnemonic	Argument List	Description
PDIST	pdist	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$	8 8-bit components, distance between

The three-dimensional array addressing instructions convert three-dimensional fixed-point addresses (in *rs1*) to a blocked-byte address.The result is stored in *rd*.

Table E-22 SPARC V-9 Three-Dimensional Array Addressing

SPARC	Mnemonic	Argument List	Description
ARRAY8	array8	reg_{rs1}, reg_{rs2}, reg_{rd}	convert 8-bit 3-D address to blocked byte address
ARRAY16	array16	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, but 16-bit
ARRAY32	array32	reg_{rs1}, reg_{rs2}, reg_{rd}	same as above, but 32-bit

Memory Access Instructions

These memory access instructions are part of the SPARC-V9 instruction set extensions.

Table E-23 SPARC-V9 Partial Store

SPARC	imm_asi	Argument List	Description
			eight 8-bit conditional stores to:
STDFA	ASI_PST8_P	stda *freg*$_{rd}$, [*freg*$_{rs1}$] *reg*$_{mask}$, *imm_asi*	primary address space
STDFA	ASI_PST8_S		secondary address space
STDFA	ASI_PST8_PL		primary address space, little endian
STDFA	ASI_PST8_SL		secondary address space, little endian
			four 16-bit conditional stores to:
STDFA	ASI_PST16_P		primary address space
STDFA	ASI_PST16_S		secondary address space
STDFA	ASI_PST16_PL		primary address space, little endian
STDFA	ASI_PST16_SL		secondary address space, little endian
			two 32-bit conditional stores to:
STDFA	ASI_PST32_P		primary address space
STDFA	ASI_PST32_S		secondary address space
STDFA	ASI_PST32_PL		primary address space, little endian
STDFA	ASI_PST32_SL		secondary address space, little endian

Note – To select a partial store instruction, use one of the partial store ASIs with the STDA instruction.

Table E-24 SPARC-V9 Short Floating-Point Load and Store

SPARC	imm_asi	Argument List	Description
			8-bit load/store from/to:
LDDFA STDFA	ASI_FL8_P	ldda [*reg_addr*] *imm_asi*, $freq_{rd}$ stda $freq_{rd}$, [*reg_addr*] *imm_asi*	primary address space
LDDFA STDFA	ASI_FL8_S	ldda [*reg_plus_imm*] %asi, $freq_{rd}$ stda [*reg_plus_imm*] %asi	secondary address space
LDDFA STDFA	ASI_FL8_PL		primary address space, little endian
LDDFA STDFA	ASI_FL8_SL		secondary address space, little endian
			16-bit load/store from/to:
LDDFA STDFA	ASI_FL16_P		primary address space
LDDFA STDFA	ASI_FL16_S		secondary address space
LDDFA STDFA	ASI_FL16_PL		primary address space, little endian
LDDFA STDFA	ASI_FL16_SL		secondary address space, little endian

Note – To select a short floating-point load and store instruction, use one of the short ASIs with the LDDA and STDA instructions

Table E-25 SPARC-V9 Atomic Quad Load

SPARC	imm_asi	Argument List	Description
LDDA	ASI_NUCLEUS_ QUAD_LDD	[*reg_addr*] *imm_asi*, reg_{rd}	128-bit atomic load
LDDA	ASI_NUCLEUS_ QUAD_LDD_L	[*reg_plus_imm*] %asi, reg_{rd}	128-bit atomic load, little endian

Table E-26 SPARC-V9 Block Load and Store

SPARC	imm_asi	Argument List	Description
			64-byte block load/store from/to:
LDDFA STDFA	ASI_BLK_AIUP	ldda [*reg_addr*] *imm_asi*, $freq_{rd}$ stda $freq_{rd}$, [*reg_addr*] *imm_asi*	primary address space, user privilege
LDDFA STDFA	ASI_BLK_AIUS	ldda [*reg_plus_imm*] %asi, $freq_{rd}$ stda $freg_{rd}$, [*reg_plus_imm*] %asi	secondary address space, user privilege.
LDDFA STDFA	ASI_BLK_AIUPL		primary address space, user privilege, little endian
LDDFA STDFA	ASI_BLK_AIUSL		secondary address space, user privilege little endian
LDDFA STDFA	ASI_BLK_P		primary address space
LDDFA STDFA	ASI_BLK_S		secondary address space
LDDFA STDFA	ASI_BLK_PL		primary address space, little endian
LDDFA STDFA	ASI_BLK_SL		secondary address space, little endian
LDDFA STDFA	ASI_BLK_COMMIT_P		64-byte block commit store to primary address space
LDDFA STDFA	ASI_BLK_COMMIT_S		64-byte block commit store to secondary address space

Note – To select a block load and store instruction, use one of the block transfer ASIs with the LDDA and STDA instructions.