Princeton University
COS 217: Introduction to Programming Systems
SPARC Assembly Language Summary

Abbreviations Used in Instruction Descriptions	
rs	Source register
rd	Destination register
ris	Source register, or Source immediate constant represented in machine language using 13 bits
is	Source immediate constant represented in machine language using 13 bits
is22	Source immediate constant represented in machine language using 22 bits
is32	Source immediate constant represented in machine language using 32 bits
addr	Memory address expressed in one of these formats: rs rs + ris rs – is is + rs is
label	Label represented in machine language as a 22-bit displacement relative to %pc
label30	Label represented in machine language as a 30-bit displacement relative to %pc
Z	Zero condition code
N	Negative condition code
V	oVerflow condition code
C	Carry condition code
r[31]	Bit 31 of r

Load and StoreInstructions (Format 3)	
ldub [addr], rd	Load an unsigned byte from addr into rd
ldsb [addr], rd	Load a signed byte from addr into rd
lduh [addr], rd	Load an unsigned halfword from addr into rd
ldsh [addr], rd	Load a signed halfword from addr into rd
ld [addr], rd	Load a word from addr into rd
ldd [addr], rd	Load a doubleword from addr and addr+1 into rd and rd+1
swap [addr], rd	Swap the contents of addr and rd
stb rs, [addr]	Store a byte from rs into addr
sth rs, [addr]	Store a halfword from rs into addr
st rs, [addr]	Store a word from rs into addr
std rs, [addr]	Store a doubleword from rs and rs+1 into addr and addr+1
clrb [addr]	Synthetic instruction for: stb %g0, addr
clrh [addr]	Synthetic instruction for: sth %g0, addr
clr [addr]	Synthetic instruction for: st %g0, addr

Shift Instructions (Format 3)	
sll rs, ris, rd	rd = rs << ris
srl rs, ris, rd	rd = rs >> ris (fill with zeros)
sra rs, ris, rd	rd = rs >> ris (fill by extending sign)

Arithmetic Instructions (Format 3)

`add rs, ris, rd`	`rd =  rs+ ris`

`addcc rs, ris, rd`	`rd = rs + ris`
	`N = rd[31] == 1`
	`Z = rd == 0`
	`V = (rs[31] & ris[31] & ~rd[31]) | (~rs[31] & ~ris[31] & rd[31])`
	`C = (rs[31] & ris[31]) | ( ~rd[31] & (rs[31] | ris[31]))`

`addx rs, ris, rd`	`rd = rs - ris + C`

`addxcc rs, ris, rd`	`rd = rs + ris + C`
	`N = rd[31] == 1`
	`Z = rd == 0`
	`V = (rs[31] & ris[31] & ~rd[31]) | (~rs[31] & ~ris[31] & rd[31])`
	`C = (rs[31] & ris[31]) | (~rd[31] & (rs[31] | ris[31]))`

`sub rs, ris, rd`	`rd = rs - ris`

`subcc rs, ris, rd`	`rd = rs - ris`
	`N = rd[31] == 1`
	`Z = rd == 0`
	`V = (rs[31] & ~ris[31] & ~rd[31]) | (~rs[31] & ris[31] & rd[31])`
	`C = (~rs[31] & ris[31]) | (rd[31] & (~rs[31] | ris[31]))`

`subx rs, ris, rd`	`rd = rs - ris - C`

`subxcc rs, ris, rd`	`rd = rs - ris - C`
	`N = rd[31] == 1`
	`Z = rd == 0`
	`V = (rs[31] & ~ris[31] & ~rd[31]) | (~rs[31] & ris[31] & rd[31])`
	`C = (~rs[31] & ris[31) | (rd[31] & (~rs[31] | ris[31]))`

`neg rs, rd`	`Synthetic instruction for:  sub %g0, rs, rd`
`neg rd`	`Synthetic instruction for:  sub %g0, rd, rd`
`inc rd`	`Synthetic instruction for:  add rd, 1, rd`
`inc is, rd`	`Synthetic instruction for:  add rd, is, rd`
`inccc rd`	`Synthetic instruction for:  addcc rd, 1, rd`
`inccc is, rd`	`Synthetic instruction for:  addcc rd, is, rd`
`dec rd`	`Synthetic instruction for:  sub rd, 1, rd`
`dec is, rd`	`Synthetic instruction for:  sub rd, is, rd`
`deccc rd`	`Synthetic instruction for:  subcc rd, 1, rd`
`deccc is, rd`	`Synthetic instruction for:  subcc rd, is, rd`
`cmp rs, ris`	`Synthetic instruction for:  subcc rs, ris, %g0`

Logical Instructions (Format 3)

`and rs, ris, rd`	`rd = rs & ris`
`andcc rs, ris, rd`	`rd = rs & ris; N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`andn rs, ris, rd`	`rd = rs & ~ris`
`andncc rs, ris, rd`	`rd = rs & ~ris; N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`or rs, ris, rd`	`rd = rs | ris`
`orcc rs, ris, rd`	`rd = rs | ris; N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`orn rs, ris, rd`	`rd = rs | ~ris`
`orncc rs, ris, rd`	`rd = rs | ~ris; N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`xor rs, ris, rd`	`rd = rs ^ ris`
`xorcc rs, ris, rd`	`rd = rs ^ ris; N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`xnor rs, ris, rd`	`rd = ~(rs ^ ris)`
`xnorcc rs, ris, rd`	`rd = ~(rs ^ ris); N = rd[31] == 1; Z = rd == 0; V = 0; C = 0`
`clr rd`	`Synthetic instruction for:  or %g0, %g0, rd`
`mov ris, rd`	`Synthetic instruction for:  or %g0, ris, rd`
`tst rs`	`Synthetic instruction for:  orcc %g0, rs, %g0`
`btst ris, rs`	`Synthetic instruction for:  andcc rs, ris, %g0`
`bset ris, rd`	`Synthetic instruction for:  or rd, ris, rd`
`bclr ris, rd`	`Synthetic instruction for:  andn rd, ris, rd`
`btog ris, rd`	`Synthetic instruction for:  xor rd, ris, rd`
`not rs, rd`	`Synthetic instruction for:  xnor rs, %g0, rd`
`not rd`	`Synthetic instruction for:  xnor rd, %g0, rd`

Integer Branch Instructions (Format 2)

	Unconditional branching:
ba{,a} label	Branch to label always
bn{,a} label	Branch to label never
	Signed number branching:
bl{,a} label	Branch to label if N ^ V
ble{,a} label	Branch to label if Z | (N ^ V)
bge{,a} label	Branch to label if ~(N ^ V)
bg{,a} label	Branch to label if ~(Z | (N ^ V))
	Unsigned number branching:
blu{,a} label	Synonym for: bcs{,a} label
bleu{,a} label	Branch to label if C | Z
bgeu{,a} label	Synonym for: bcc{,a} label
bgu{,a} label	Branch to label if ~(C | Z)
	Individual condition code branching:
be{,a} label	Branch to label if Z
bne{,a} label	Branch to label if ~Z
bpos{,a} label	Branch to label if ~N
bneg{,a} label	Branch to label if N
bcs{,a} label	Branch to label if C
bcc{,a} label	Branch to label if ~C
bvs{,a} label	Branch to label if V
bvc{,a} label	Branch to label if ~V
bz{,a} label	Synonym for: be{,a} label
bnz{,a} label	Synonym for: bne{,a} label

Control Instructions (Format 3)

jmpl addr, rd	Store %pc in rd, and jump to addr
jmp addr	Synthetic instruction for: jmpl addr, %g0
call ris	Synthetic instruction for: jmpl ris, %o7
ret	Synthetic instruction for: jmpl %i7 + 8, %g0
retl	Synthetic instruction for: jmpl %o7 + 8, %g0
save rs, ris, rd	Save register window. rd = rs + ris
restore rs, ris, rd	Restore register window. rd = rs + ris
restore	Synthetic instruction for: restore %g0, %g0, %g0

Control Instructions (Format 2)

nop	No operation
sethi is22, rd	Set the high-order 22 bits of rd to is22, and set the low-order 10 bits of rd to 0

set is32, rd	Synthetic instruction for:
	  sethi %hi(is32), rd
	  or rd, %lo(is32), rd

Control Instructions (Format 1)

call label30	Store %pc in %o7, and jump to label30

Trap Instructions (Format 3)

ta addr	Trap always to addr (typically 0)
...	

Floating-Point Instructions (Format 3)

...	

Pseudo-Ops	
symbol:	Define a label named symbol whose value is the current location counter.
symbol = expr	Define an assembler constant. The assembler replaces symbol with the value of expr.
.section ".text"	Add the following code to the text section. The text section contains executable code.
.section ".data"	Add the following code to the data section. The data section contains program-initialized read-write data.
.section ".bss"	Add the following code to the bss section. The bss section contains read-write data that is initialized to 0.
.section ".rodata"	Add the following code to the rodata section. The rodata contains read-only data.
.skip n	Skip n bytes of memory.
.align n	Increase the location counter so its value is evenly divisible by n.
.byte bytevalue1, bytevalue2, ...	Allocate memory containing bytevalue1, bytevalue2, ...
.half halfvalue1, halfvalue2, ...	Allocate memory containing halfvalue1, halfvalue2, ...
.word wordvalue1, wordvalue2, ...	Allocate memory containing wordvalue1, wordvalue2, ...
.ascii "string1", "string2", ...	Allocate memory containing the characters from string1, string2, ...
.asciz "string1", "string2", ...	Allocate memory containing string1, string2, ... where each string is NULL terminated.
.common symbol, size	Declare the name and size of a common area of memory to be shared by multiple object files.
.global symbol1, symbol2, ...	Mark symbol1, symbol2, ... so they are available to the linker.
.empty	Suppress assembler warnings about the next instruction's presence in a delay slot.