# COP5621 - Spring 2020
# Assignment 6
# cgen

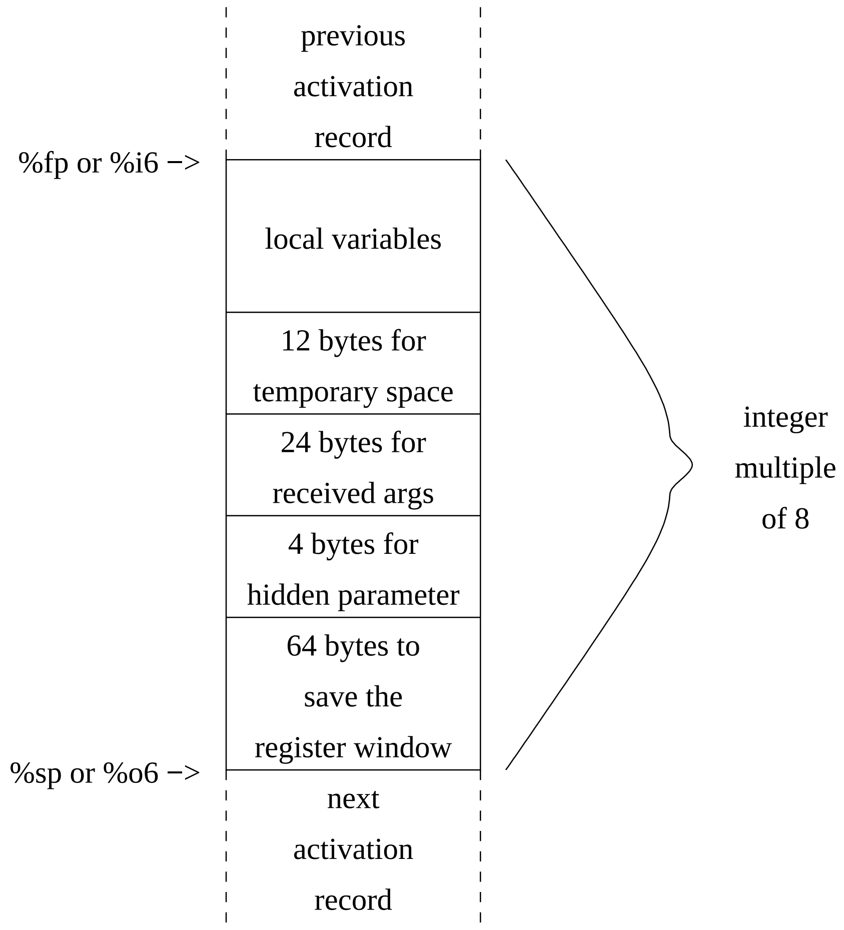
*cgen* reads *csem's* intermediate code from its standard input and compiles it into an assembly language program for the SPARC on its standard output. It naively expands each quadruple operation in a straight-forward manner (without any attempt at optimization). It uses the following conventions.

```
        string definition
        .seg    "data"
LSn:
        .asciz  "string"
        .seg    "text"

        symbol definition
Bn=Lm

        function header
        .seg    "text"
        .global f
f:
        save    %sp,(-locsize),%sp

        function call
        store into %o0
        store into %o1
        ...
        store into %o5
        call    reg containing addr of func,num args
        nop
        mov     %o0,reg

        function return
        store into %i0
        ret
        restore
```

Locals and parameters are located at `%sp + n`. Labels should be of the form `Ln`. The allocatable registers are %g1-%g7 (%1-%7) [global], %o0-%o5 (%8-%13) [output], %l0-%l7 (%16-%23) [local], %i0-%i5 (%24-%29) [input], and %f0-%f31 [floating-point]. Only %l0-%l7 and %i0-%i5 are nonscratch (retain their values across function calls). Arguments are passed to functions in the output registers and received by functions in the input registers. You may just print an error message if there are live floating-point values across calls or too many arguments for the output or input registers.

The following pages give an example C program, the intermediate code produced (which will be input to your program), and the corresponding SPARC assembly code. Also enclosed is some information about SPARC registers, activation records, and instructions. You can use the command "gcc -o *name*.exe *name*.s" to assemble and create an executable that you can test.

For this assignment, create a single file (e.g. cgen.c). E-mail this file to *whalley@cs.fsu.edu* before the beginning of class on April 16. Please put COP5621 somewhere on the subject line of the message. Make sure that you indicate how the file should be compiled in your header information. You should test your assignment 6 solution on *program.cs.fsu.edu*. You can reference my csem executable (*˜whalley/cop5621ex/csem*) on *program.cs.fsu.edu* to produce quadruples as input to your program.



Layout of an Activation Record on the SPARC

```
/*
 * demonstrate argument passing and simple arithmetic operations
 */
main()
{
   double x;
   int i;

   i = 10;
   x = 15;
   printf("i is %d\n", i);
   printf("x is %f\n", x);
   i += 3;
   x /= 3;
   foo(i, x);
}

foo(int j, double z)
{
   printf("j is %d\n", j);
   printf("z is %f\n", z);
}
```

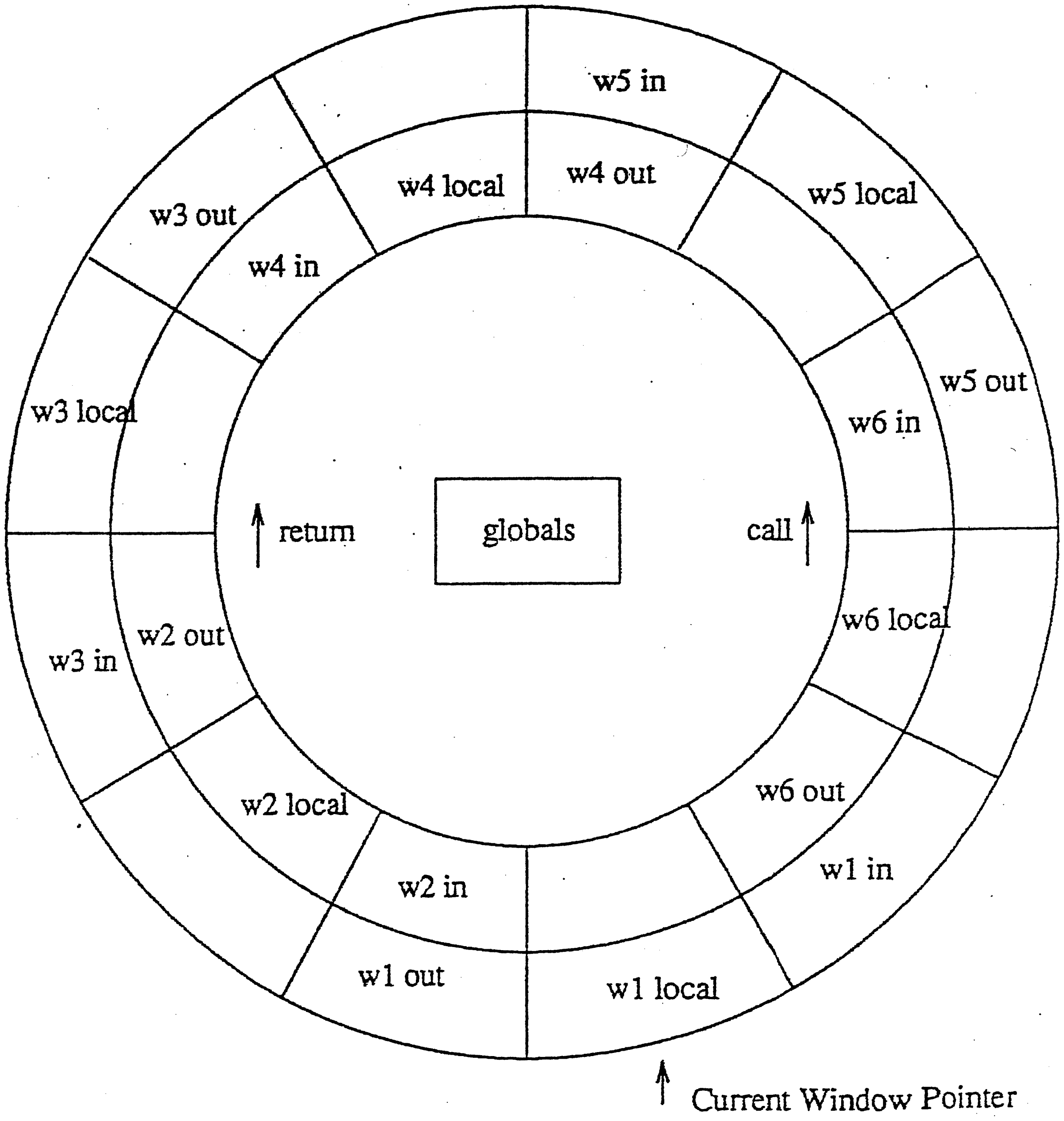
```
func main
localloc 8
localloc 4
bgnstmt 9
t1 := local 1
t2 := 10
t3 := t1 =i t2
bgnstmt 10
t4 := local 0
t5 := 15
t6 := cvf t5
t7 := t4 =f t6
bgnstmt 11
t8 := "i is %d\n"
t9 := local 1
t10 := @i t9
argi t8
argi t10
t11 := global printf
t12 := fi t11 2
bgnstmt 12
t13 := "x is %f\n"
t14 := local 0
t15 := @f t14
argi t13
argf t15
t16 := global printf
t17 := fi t16 2
bgnstmt 13
t18 := local 1
t19 := 3
t20 := @i t18
t21 := t20 +i t19
t22 := t18 =i t21
bgnstmt 14
t23 := local 0
t24 := 3
t25 := cvf t24
t26 := @f t23
t27 := t26 /f t25
t28 := t23 =f t27
bgnstmt 15
t29 := local 1
t30 := @i t29
t31 := local 0
t32 := @f t31
argi t30
argf t32
t33 := global foo
t34 := fi t33 2
fend
func foo
formal 4
formal 8
bgnstmt 20
t35 := "j is %d\n"
t36 := param 0
t37 := @i t36
argi t35
argi t37
t38 := global printf
t39 := fi t38 2
bgnstmt 21
t40 := "z is %f\n"
t41 := param 1
t42 := @f t41
argi t40
argf t42
t43 := global printf
```

```
t44 := fi t43 2
fend
```

```
        .seg    "text"
        .global main
main:
        save    %sp,(-120),%sp
        add     %sp,112,%l0
        mov     10,%l1
        st      %l1,[%l0]
        add     %sp,104,%l0
        mov     15,%l1
        st      %l1,[%sp + 96]
        ld      [%sp + 96],%f0
        fitod   %f0,%f0
        std     %f0,[%l0]
        .seg    "data"
LS8:
        .asciz  "i is %d\n"
        .seg    "text"
        sethi   %hi(LS8),%l0
        or      %l0,%lo(LS8),%l0
        add     %sp,112,%l1
        ld      [%l1],%l2
        sethi   %hi(printf),%l3
        or      %l3,%lo(printf),%l3
        mov     %l0,%o0
        mov     %l2,%o1
        call    %l3,2
        nop
        .seg    "data"
LS13:
        .asciz  "x is %f\n"
        .seg    "text"
        sethi   %hi(LS13),%l0
        or      %l0,%lo(LS13),%l0
        add     %sp,104,%l1
        ldd     [%l1],%f0
        sethi   %hi(printf),%l2
        or      %l2,%lo(printf),%l2
        mov     %l0,%o0
        std     %f0,[%sp + 96]
        ld      [%sp + 96],%o1
        ld      [%sp + 100],%o2
        call    %l2,3
        nop
        add     %sp,112,%l0
        mov     3,%l1
        ld      [%l0],%l2
        add     %l2,%l1,%l2
        st      %l2,[%l0]
        add     %sp,104,%l0
        mov     3,%l1
        ldd     [%l0],%f0
        st      %l1,[%sp + 96]
        ld      [%sp + 96],%f2
        fitod   %f2,%f2
        fdivd   %f0,%f2,%f0
        std     %f0,[%l0]
        add     %sp,112,%l0
        ld      [%l0],%l1
        add     %sp,104,%l2
        ldd     [%l2],%f0
        sethi   %hi(foo),%l3
        or      %l3,%lo(foo),%l3
        mov     %l1,%o0
        std     %f0,[%sp + 96]
        ld      [%sp + 96],%o1
        ld      [%sp + 100],%o2
        call    %l3,3
        nop
        ret
```

```
        restore
        .seg    "text"
        .global foo
foo:
        save    %sp,(-104),%sp
        st      %i0,[%fp + 68]
        st      %i1,[%fp + 72]
        st      %i2,[%fp + 76]
        .seg    "data"
LS35:
        .asciz  "j is %d\n"
        .seg    "text"
        sethi   %hi(LS35),%l0
        or      %l0,%lo(LS35),%l0
        add     %fp,68,%l1
        ld      [%l1],%l2
        sethi   %hi(printf),%l3
        or      %l3,%lo(printf),%l3
        mov     %l0,%o0
        mov     %l2,%o1
        call    %l3,2
        nop
        .seg    "data"
LS40:
        .asciz  "z is %f\n"
        .seg    "text"
        sethi   %hi(LS40),%l0
        or      %l0,%lo(LS40),%l0
        add     %fp,72,%l1
        ldd     [%l1],%f0
        sethi   %hi(printf),%l2
        or      %l2,%lo(printf),%l2
        mov     %l0,%o0
        std     %f0,[%sp + 96]
        ld      [%sp + 96],%o1
        ld      [%sp + 100],%o2
        call    %l2,3
        nop
        ret
        restore
```

Figure 2-2 *Overlapping Register Windows*



# 2 Instruction-set Mapping

The tables in this chapter describe the relationship between hardware instructions of the SPARC architecture, as defined in *SPARC Processor Architecture*, and the instruction set used by Sun Microsystems' SPARC Assembler.

## 2.1. Table Notation

The following table describes the notation used in the tables in the rest of the chapter to describe the instruction set of the assembler. In the table below, the vertical bar | indicates alternation, but brackets [ ] are to be taken literally:

Table 2-1 *Notation*

| *Symbol* | *Describes* | *Comment* |
|---|---|---|
| *reg* | `%0 ... %31`<br>`%g0 ... %g7`<br>`%o0 ... %o7`<br>`%l0 ... %l7`<br>`%i0 ... %i7` | <br>*(same as* `%0...%7`*)*<br>*(same as* `%8...%15`*)*<br>*(same as* `%16...%23`*)*<br>*(same as* `%24...%31`*)* |
| *freg* | `%f0 ... %f31` | |
| *creg* | `%c0 ... %c31` | |
| *value* | | *(an expression involving at most one relocatable symbol)* |
| *const13* | *value* | *(a signed constant which fits in 13 bits)* |
| *const22* | *value* | *(a constant which fits in 22 bits)* |
| *asi*<br><br>$reg_{rd}$ | *value* | *(alternate address space identifier; an unsigned value fitting in 8 bits)*<br>*Destination register.* |
| $reg_{rs1}$, $reg_{rs2}$ | | *Source register 1, source register 2.* |
| *regaddr* | | *Address formed with register contents only.* |
| *address* | $reg_{rs1}$ + $reg_{rs2}$<br>$reg_{rs1}$ + *const13*<br>$reg_{rs1}$ − *const13*<br>*const13* + $reg_{rs1}$<br>*const13* | *Address formed from register contents, immediate constant, or both.* |

Table 2-1 *Notation— Continued*

| *Symbol* | *Describes* | *Comment* |
|---|---|---|
| *reg_or_imm* | *reg*$_{rs2}$ *const13* | *Value from either a single register, or an immediate constant.* |

## 2.2. Hardware Integer Instructions

The following table outlines the correspondence between SPARC hardware instructions and SPARC assembly-language instructions. The following notations are suffixed repeatedly to assembler mnemonics (and in upper case for SPARC architecture instruction names):

*sr* — for status register.

*a* — for instructions dealing with alternate space.

*b* — for byte instructions.

*h* — for halfword instructions.

*d* — for double-word instructions.

*f* — for referencing floating-point registers.

*c* — for referencing coprocessor registers.

*rd* — as a subscript, refers to a destination register in the argument list of an instruction.

*rs* — as a subscript, refers to a source register in the argument list of an instruction.

NOTE *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 *All* `Bicc` *and* `Bfcc` *instructions, described in the following table, may indicate that the* annul *bit is to be set by appending* "`,a`" *to the opcode; e.g.* "`bgeu,a` *label*". The following syntax descriptions indicate the optional "`,a`" *by enclosing it in* { }:

Table 2-2 *SPARC to Assembly Language Mapping*

| *SPARC* | *Mnemonic* | *Argument List* | *Name* | *Comments* |
|---|---|---|---|---|
| LDSB | `ldsb` | [*address*], *reg*$_{rd}$ | *Load signed byte.* | |
| LDSH | `ldsh` | [*address*], *reg*$_{rd}$ | *Load signed halfword.* | |
| LDUB | `ldub` | [*address*], *reg*$_{rd}$ | *Load unsigned byte.* | |
| LDUH | `lduh` | [*address*], *reg*$_{rd}$ | *Load unsigned halfword.* | |
| LD | `ld` | [*address*], *reg*$_{rd}$ | *Load word.* | |
| D | `ldd` | [*address*], *reg*$_{rd}$ | *Load double word.* | (*reg*$_{rd}$ *must be even*) |

Table 2-2 *SPARC to Assembly Language Mapping— Continued*

| *SPARC* | *Mnemonic* | *Argument List* | *Name* | *Comments* |
|---|---|---|---|---|
| LDF | `ld` | [*address*], $freg_{rd}$ | *Load floating-point register.* | |
| LDFSR | `ld` | [*address*], `%fsr` | | |
| LDDF | `ldd` | [*address*], $freg_{rd}$ | *Load double floating-point.* | |
| LDC | `ld` | [*address*], $creg_{rd}$ | *Load coprocessor.* | |
| LDCSR | `ld` | [*address*], `%csr` | | |
| LDDC | `ldd` | [*address*], $creg_{rd}$ | *Load double coprocessor.* | |
| 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)* |
| STB | `stb` | *regaddr*, [*address*] | *Store byte.* | *(synonyms:* `stub`, `stsb`*)* |
| STH | `sth` | *regaddr*, [*address*] | | *(synonyms:* `stuh`, `stsh`*)* |
| ST | `st` | $reg_{rd}$, [*address*] | | |
| STD | `std` | $reg_{rd}$, [*address*] | | *(*$reg_{rd}$ *must be even)* |
| STF | `st` | $freg_{rd}$, [*address*] | | |
| STDF | `std` | $freg_{rd}$, [*address*] | | |
| STFSR | `st` | `%fsr`, [*address*] | *Store floating-point status register.* | |
| STDFQ | `std` | `%fq`, [*address*] | *Store double floating-point queue.* | |
| STC | `st` | $creg_{rd}$, [*address*] | *Store coprocessor.* | |
| STDC | `std` | $creg_{rd}$, [*address*] | | |
| STCSR | `st` | `%csr`, [*address*] | | |
| STDCQ | `std` | `%cq`, [*address*] | *Store double coprocessor queue.* | |
| STBA | `stba` | *regaddr*, [*regaddr*] *asi* | *Store byte into alternate space.* | *(synonyms:* `stuba`, `stsba`*)* |
| STHA | `stha` | *regaddr*, [*regaddr*] *asi* | | *(synonyms:* `stuha`, `stsha`*)* |
| STA | `sta` | $reg_{rd}$, [*regaddr*] *asi* | | |
| STDA | `stda` | $reg_{rd}$, [*regaddr*] *asi* | | *(*$reg_{rd}$ *must be even)* |
| LDSTUB | `ldstub` | [*address*], $reg_{rd}$ | *Load-store unsigned byte.* | |
| LDSTUBA | `ldstuba` | [*regaddr*] *asi*, $reg_{rd}$ | | |
| SWAP | `swap` | [*address*], $reg_{rd}$ | *Swap memory word with register.* | |
| SWAPA | `swapa` | [*regaddr*] *asi*, $reg_{rd}$ | | |

Table 2-2 *SPARC to Assembly Language Mapping— Continued*

| *SPARC* | *Mnemonic* | *Argument List* | *Name* | *Comments* |
|---|---|---|---|---|
| ADD | `add` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Add* | |
| ADDcc | `addcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Add and modify icc.* | |
| ADDX | `addx` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Add with carry.* | |
| ADDXcc | `addxcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| SUB | `sub` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Subtract.* | |
| SUBcc | `subcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Subtract and modify icc.* | |
| SUBX | `subx` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Subtract with carry.* | |
| SUBXcc | `subxcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| MULScc | `mulscc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Multiply step (and modify icc).* | |
| AND | `and` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *And.* | |
| ANDcc | `andcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| ANDN | `andn` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| ANDNcc | `andncc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| OR | `or` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Inclusive or.* | |
| ORcc | `orcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| ORN | `orn` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| RNcc | `orncc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| XOR | `xor` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Exclusive or.* | |
| XORcc | `xorcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| XNOR | `xnor` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Exclusive nor.* | |
| XNORcc | `xnorcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| TADDcc | `taddcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Tagged add.* | |
| TSUBcc | `tsubcc` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| TADDccTV | `taddcctv` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Tagged add and modify icc and trap on overflow.* | |
| TSUBccTV | `tsubcctv` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | | |
| SLL | `sll` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Shift left logical.* | |
| SRL | `srl` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Shift right logical.* | |
| SRA | `sra` | $reg_{rs2}$, *reg_or_imm*, $reg_{rd}$ | *Shift right arithmetic.* | |
| SETHI | `sethi` | *const22*, $reg_{rd}$ | *Set high 22 bits of r register.* | |
| | `sethi` | `%hi(`*value*`)`, $reg_{rd}$ | | *(see pseudo-instructions)* |
| SAVE | `save` | $reg_{rs1}$, *reg_or_imm*, $reg_{rd}$ | | *(see pseudo-instructions)* |
| RESTORE | `restore` | $reg_{rs1}$, *reg_or_imm*, $reg_{rd}$ | | *(see pseudo-instructions)* |
| Bicc | `bn{,a}` | *label* | *Branch on integer condition codes..* | *(branch never)* |
| | `bne{,a}` | *label* | | *(synonym:* `bnz`*)* |
| | `be{,a}` | *label* | | *(synonym:* `bz`*)* |
| | `bg{,a}` | *label* | | |

Table 2-2 *SPARC to Assembly Language Mapping— Continued*

| *SPARC* | *Mnemonic* | *Argument List* | *Name* | *Comments* |
|---|---|---|---|---|
| Bicc | `ble{,a}` | *label* | | |
| | `bge{,a}` | *label* | | |
| | `bl{,a}` | *label* | | |
| | `bgu{,a}` | *label* | | |
| | `bleu{,a}` | *label* | | |
| | `bcc{,a}` | *label* | | *(synonym:* `bgeu`*)* |
| | `bcs{,a}` | *label* | | *(synonym:* `blu`*)* |
| | `bpos{,a}` | *label* | | |
| | `bneg{,a}` | *label* | | |
| | `bvc{,a}` | *label* | | |
| | `bvs{,a}` | *label* | | |
| | `ba{,a}` | *label* | | *(synonym:* `b`*)* |
| FBfcc | `fbn{,a}` | *label* | *Branch on floating-point condition codes.* | *(branch never)* |
| | `fbu{,a}` | *label* | | |
| | `fbg{,a}` | *label* | | |
| | `fbug{,a}` | *label* | | |
| | `fbl{,a}` | *label* | | |
| | `fbul{,a}` | *label* | | |
| | `fblg{,a}` | *label* | | |
| | `fbne{,a}` | *label* | | *(synonym:* `fbnz`*)* |
| | `fbe{,a}` | *label* | | *(synonym:* `fbz`*)* |
| | `fbue{,a}` | *label* | | |
| | `fbge{,a}` | *label* | | |
| | `fbuge{,a}` | *label* | | |
| | `fble{,a}` | *label* | | |
| | `fbule{,a}` | *label* | | |
| | `fbo{,a}` | *label* | | |
| | `fba{,a}` | *label* | | |
| CBccc | `cbn{,a}` | *label* | *Branch on coprocessor condition codes.* | *(branch never)* |
| | `cb3{,a}` | *label* | | |
| | `cb2{,a}` | *label* | | |
| | `cb23{,a}` | *label* | | |
| | `cb1{,a}` | *label* | | |
| | `cb13{,a}` | *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 2-2 *SPARC to Assembly Language Mapping— Continued*

| *SPARC* | *Mnemonic* | *Argument List* | *Name* | *Comments* |
|---|---|---|---|---|
| CALL | `call` | *label* [, *n*] | (*n = # of out registers used as arguments*) | |
| UNIMP | `unimp` | *const22* | *Unimplemented instruction.* | |
| JMPL | `jmpl` | *address*, *reg*$_{rd}$ | *Jump and link.* | |
| RETT | `rett` | *address* | *Return from trap.* | |
| IFLUSH | `iflush` | *address* | *Instruction cache flush.* | |
| Ticc | `tn` | *address* | *Trap on integer condition code. (See note.)* | *(trap never)* |
| | `tne` | *address* | | *(synonym:* `tnz`*)* |
| | `te` | *address* | | *(synonym:* `tz`*)* |
| | `tg` | *address* | | |
| | `tle` | *address* | | |
| | `tge` | *address* | | |
| | `tl` | *address* | | |
| | `tgu` | *address* | | |
| | `tleu` | *address* | | |
| | `tlu` | *address* | | *(synonym:* `tcc`*)* |
| | `tgeu` | *address* | | *(synonym:* `tcs`*)* |
| | `tpos` | *address* | | |
| | `tneg` | *address* | | |
| | `tvc` | *address* | | |
| | `tvs` | *address* | | |
| | `ta` | *address* | | *(synonym:* `t`*)* |
| RDY | `rd` | `%y`, *reg*$_{rd}$ | | *(see pseudo-instructions)* |
| RDPSR | `rd` | `%psr`, *reg*$_{rd}$ | | *(see pseudo-instructions)* |
| RDWIM | `rd` | `%wim`, *reg*$_{rd}$ | | *(see pseudo-instructions)* |
| RDTBR | `rd` | `%tbr`, *reg*$_{rd}$ | | *(see pseudo-instructions)* |
| WRY | `wr` | *reg*$_{rs1}$, *reg_or_imm*, `%y` | | *(see pseudo-instructions)* |
| WRPSR | `wr` | *reg*$_{rs1}$, *reg_or_imm*, `%psr` | | *(see pseudo-instructions)* |
| WRWIM | `wr` | *reg*$_{rs1}$, *reg_or_imm*, `%wim` | | *(see pseudo-instructions)* |
| WRTBR | `wr` | *reg*$_{rs1}$, *reg_or_imm*, `%tbr` | | *(see pseudo-instructions)* |

*NOTE* *Trap numbers 16-31 are available for use by the user, and will not be usurped by Sun. Currently-defined trap numbers are those defined in* `/usr/include/sun4/trap.h`, *as follows:*

| | |
|---|---|
| `0x00` | `ST_SYSCALL` |
| `0x01` | `ST_BREAKPOINT` |
| `0x02` | `ST_DIV0` |
| `0x03` | `ST_FLUSH_WINDOWS` |

```
0x04  ST_CLEAN_WINDOWS

0x05  ST_RANGE_CHECK
```

## 2.3. Hardware Floating-Point Instructions

In the table below, the types of numbers being manipulated by an instruction are denoted by the following lowercase letters:

*i* — integer

*s* — single

*d* — double

*x* — extended

In some 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 2-3 *Floating-point Instructions*

| *SPARC* | *Mnemonic* | *Argument List* | *Description* |
|---|---|---|---|
| `FiTOs` | `fitos` | $freg_{rs2}$, $freg_{rd}$ | *Convert integer to single.* |
| `FiTOd` | `fitod` | $freg_{rs2}$, $freg_{rd}$ | *Convert integer to double.* |
| `iTOx` | `fitox` | $freg_{rs2}$, $freg_{rd}$ | *Convert integer to extended.* |
| `sTOi` | `fstoi` | $freg_{rs2}$, $freg_{rd}$ | *Convert single to integer.* |
| `FdTOi` | `fdtoi` | $freg_{rs2}$, $freg_{rd}$ | *Convert double to integer.* |
| `FxTOi` | `fxtoi` | $freg_{rs2}$, $freg_{rd}$ | *Convert extended to integer.* |
| `FINTs` | `fints` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued single.* |
| `FINTd` | `fintd` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued double.* |
| `FINTx` | `fintx` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued extended.* |
| `FINTRZs` | `fintrzs` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued single & round toward zero.* |
| `FINTRZd` | `fintrzd` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued double & round toward zero.* |
| `FINTRZx` | `fintrzx` | $freg_{rs2}$, $freg_{rd}$ | *Convert to integral-valued extended & round toward zero.* |
| `FsTOd` | `fstod` | $freg_{rs2}$, $freg_{rd}$ | *Convert single to double.* |
| `FsTOx` | `fstox` | $freg_{rs2}$, $freg_{rd}$ | *Convert single to extended.* |
| `FdTOs` | `fdtos` | $freg_{rs2}$, $freg_{rd}$ | *Convert double to single.* |
| `FdTOx` | `fdtox` | $freg_{rs2}$, $freg_{rd}$ | *Convert double to extended.* |
| `FxTOd` | `fxtod` | $freg_{rs2}$, $freg_{rd}$ | *Convert extended to double.* |
| `FxTOs` | `fxtos` | $freg_{rs2}$, $freg_{rd}$ | *Convert extended to single.* |
| `FMOVs` | `fmovs` | $freg_{rs2}$, $freg_{rd}$ | *move* |
| `FNEGs` | `fnegs` | $freg_{rs2}$, $freg_{rd}$ | *negate* |
| `FABSs` | `fabss` | $freg_{rs2}$, $freg_{rd}$ | *absolute value* |
| `CLASSs` | `fclasss` | $freg_{rs2}$, $freg_{rd}$ | *classify* |
| `CLASSd` | `fclassd` | $freg_{rs2}$, $freg_{rd}$ | |

Table 2-3 *Floating-point Instructions— Continued*

| *SPARC* | *Mnemonic* | *Argument List* | *Description* |
|---|---|---|---|
| FCLASSx | fclassx | $freg_{rs2}, freg_{rd}$ | |
| FEXPOs | fexpos | $freg_{rs2}, freg_{rd}$ | *extract exponent* |
| FEXPOd | fexpod | $freg_{rs2}, freg_{rd}$ | |
| FEXPOx | fexpox | $freg_{rs2}, freg_{rd}$ | |
| FSQRTs | fsqrts | $freg_{rs2}, freg_{rd}$ | *square root* |
| FSQRTd | fsqrtd | $freg_{rs2}, freg_{rd}$ | |
| FSQRTx | fsqrtx | $freg_{rs2}, freg_{rd}$ | |
| FADDs | fadds | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | *add* |
| FADDd | faddd | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FADDx | faddx | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FSUBs | fsubs | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | *subtract* |
| FSUBd | fsubd | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FSUBx | fsubx | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FMULs | fmuls | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | *multiply* |
| FMULd | fmuld | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FMULx | fmulx | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FDIVs | fdivs | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | *divide* |
| IVd | fdivd | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| DIVx | fdivx | $freg_{rs1}, freg_{rs2}, freg_{rd}$ | |
| FSCALEs | fscales | $freg_{rs2}, freg_{rd}$ | *scale* |
| FSCALEd | fscaled | $freg_{rs2}, freg_{rd}$ | |
| FSCALEx | fscalex | $freg_{rs2}, freg_{rd}$ | |
| FREMs | frems | $freg_{rs2}, freg_{rd}$ | *partial remainder* |
| FREMd | fremd | $freg_{rs2}, freg_{rd}$ | |
| FREMx | fremx | $freg_{rs2}, freg_{rd}$ | |
| FQUOTs | fquots | $freg_{rs2}, freg_{rd}$ | *partial quotient* |
| FQUOTd | fquotd | $freg_{rs2}, freg_{rd}$ | |
| FQUOTx | fquotx | $freg_{rs2}, freg_{rd}$ | |
| FCMPs | fcmps | $freg_{rs1}, freg_{rs2}$ | *compare* |
| FCMPd | fcmpd | $freg_{rs1}, freg_{rs2}$ | |
| FCMPx | fcmpx | $freg_{rs1}, freg_{rs2}$ | |
| FCMPEs | fcmpes | $freg_{rs1}, freg_{rs2}$ | *Compare, generate exception if unordered.* |
| FCMPEd | fcmped | $freg_{rs1}, freg_{rs2}$ | |
| FCMPEx | fcmpex | $freg_{rs1}, freg_{rs2}$ | |

## 2.4. Pseudo-Instructions

This section describes the mapping of pseudo-instructions to hardware instructions.

Table 2-4 *Pseudo-Instruction to Hardware Instruction Mapping*

| *Pseudo-Instruction* | | *Hardware Equivalent(s)* | | *Comment* |
|---|---|---|---|---|
| `nop` | | `sethi` | `0,%g0` | *(no-op)* |
| `cmp` | $reg_{rs1}$, *reg_or_imm* | `subcc` | $reg_{rs1}$, *reg_or_imm*, `%g0` | *(compare)* |
| `jmp` | *address* | `jmpl` | *address*, `%g0` | |
| `call` | *reg_or_imm* | `jmpl` | *reg_or_imm*, `%o7` | |
| `tst` | $reg_{rs1}$ | `orcc` | $reg_{rs1}$, `%g0,%g0` | *(test)* |
| `ret`<br>`retl`<br>`restore`<br>`save` | | `jmpl`<br>`jmpl`<br>`restore`<br>`save` | `%i7+8,%g0`<br>`%o7+8,%g0`<br>`%g0,%g0,%g0`<br>`%g0,%g0,%g0` | *(return from subroutine)*<br>*(return from leaf subroutine)*<br>*(trivial* `restore`*)*<br>*(trivial* `save`*)*<br>*Warning: trivial* `save` *should only be used in kernel code!* |
| `set`<br>`set`<br>`set` | *value*, $reg_{rd}$<br>*value*, $reg_{rd}$<br>*value*, $reg_{rd}$ | `or`<br>`sethi`<br>`sethi`<br>`or` | `%g0`, *value*, $reg_{rd}$<br>`%hi(`*value*`)`, $reg_{rd}$<br>`%hi(`*value*`)`, $reg_{rd}$;<br>$reg_{rd}$, `%lo(`*value*`)`, $reg_{rd}$ | *(if -4096 ≤ value ≤ 4095)*<br>*(if ((value&0x1ff) == 0))*<br>*(otherwise)*<br><br>*Warning: do not use* `set` *in an instruction's delay slot.* |
| `not`<br>`not`<br>`neg`<br>`neg` | $reg_{rs1}$, $reg_{rd}$<br>$reg_{rd}$<br>$reg_{rs2}$, $reg_{rd}$<br>$reg_{rd}$ | `xnor`<br>`xnor`<br>`sub`<br>`sub` | $reg_{rs1}$, `%g0`, $reg_{rd}$<br>$reg_{rd}$, `%g0`, $reg_{rd}$<br>`%g0`, $reg_{rs2}$, $reg_{rd}$<br>`%g0`, $reg_{rd}$, $reg_{rd}$ | *(one's complement)*<br>*(one's complement)*<br>*(two's complement)*<br>*(two's complement)* |
| `inc`<br>`inc`<br>`inccc`<br>`inccc` | $reg_{rd}$<br>*const13*, $reg_{rd}$<br>$reg_{rd}$<br>*const13*, $reg_{rd}$ | `add`<br>`add`<br>`addcc`<br>`addcc` | $reg_{rd}$, `1`, $reg_{rd}$<br>$reg_{rd}$, *const13*, $reg_{rd}$<br>$reg_{rd}$, `1`, $reg_{rd}$<br>$reg_{rd}$, *const13*, $reg_{rd}$ | *(increment by 1)*<br>*(increment by const13)*<br>*(increment by 1 and set icc)*<br>*(increment by const13 and set icc)* |
| `dec`<br>`dec`<br>`deccc`<br>`deccc` | $reg_{rd}$<br>*const13*, $reg_{rd}$<br>$reg_{rd}$<br>*const13*, $reg_{rd}$ | `sub`<br>`sub`<br>`subcc`<br>`subcc` | $reg_{rd}$, `1`, $reg_{rd}$<br>$reg_{rd}$, *const13*, $reg_{rd}$<br>$reg_{rd}$, `1`, $reg_{rd}$<br>$reg_{rd}$, *const13*, $reg_{rd}$ | *(decrement by 1)*<br>*(decrement by const13)*<br>*(decrement by 1 and set icc)*<br>*(decrement by const13 and set icc)* |