

SPARC

~ 32 general-purpose registers
Only simple load/store to memory

```
ld [%r2], %r3  
ld [%r2 + 12], %r3  
ld [%r2 + %r7], %r3 ; 13 bit signed offset
```

All arithmetic/logical ops are register-to-register

```
add %r2, %r1, %r1  
add %r2, 17, %r5 ; 13 bit signed immediate
```

Ops set condition codes, which are tested by **Bicc** ops, e.g.

```
bne label ; branch if last op result <> 0  
ble label ; branch if last op result <= 0
```

Special instruction to load high-order bits of full-word constant

```
sethi %hi(value), %r7 ; net effect is to load  
or %r7,%lo(value), %r7 ; all of `value` into r7
```

12/31/04

PSU CS322 W05(C) Andrew Tolmach 1997-2005

3

Introduction to SPARC Architecture

SPARC = **S**caleable **P**rocessor **ARC**hitecture

One of original RISC machines

- RISC = **R**educed **I**nstruction **S**et **C**omputer
- Developed from research at U.C.-Berkeley ca. 1980
- Commercialized by Sun Microsystems
- Licensed freely to others.
- Similar, parallel effort at Stanford led to MIPS machine
- Also IBM RS/6000, HP/PA, DEC ALPHA, PowerPC, etc.

Key idea: simple uniform instruction set allowing **fast** cycle times (and fast time to market)

Has eclipsed CISC machines (C = Complex) like the DEC VAX, but not the Intel x86 line.

V8 is 32-bit version; V9 is 64-bit extension with extra instructions. We'll assume V8 in this course.

12/31/04

PSU CS322 W05(C) Andrew Tolmach 1997-2005

2

SPARC - Delayed Branches

Implementation is **pipelined**.

It takes time to process a branch, because must fetch and decode target instruction

So why not execute a useful instruction in the meantime?

Specify this instruction by placing it in the **delay slot**:

```
tst %r6 ; executes first  
bne L1 ; executes third  
inc %r5 ; delay slot: executes second, whether  
; or not branch is executed
```

Also applies to calls and returns:

```
call _fred  
move %9, %o0 ; executes before call  
move %o0, %i7 ; executes after call
```

Loads, stores, floating point ops may also cause delays, but programmer generally can't see these directly.

12/31/04

PSU CS322 W05(C) Andrew Tolmach 1997-2005

4

SPARC Activation Records (for C)

%sp - always points to true top of stack.

%fp - "frame pointer" - points to top of stack at entry to procedure; used to reference stack-passed arguments and stack-stored local variables.

The **save** instruction creates a stack frame, by subtracting a specified number of bytes from **%sp** and switching windows; **old %sp** becomes **new %fp**.

But in simple cases, arguments are just passed in registers (**%o0,%o1**, etc. for caller; **%i0,%i1**, etc. for callee) and locals are kept in registers too (**%l0**, etc.) so nothing important goes in frame.

Still, always need to allocate a frame of **minimum** size 96 bytes. If you store locals on the stack, you may need more. (Hint: Compiling with **-O2** will make

gcc try hard to avoid using the frame.)

12/31/04 PSU CS322 W05(C) Andrew Tolmach 1997-2005

9

SPARC Calling Conventions - Example

```
int power (int a, int n) {
    if (n > 0)
        return a * power(a,n-1);
    else
        return 1;}
```

```
.section ".text"
.align 4
.global power

power:
    save    %sp, -96, %sp
    cmp    %i1, 0
    ble,a  done
    mov    1, %i0
    add    %i1, -1, %o1
    call   power
    mov    %i0,%o0
    umul   %i0,%o0,%i0
    done:
    ret

restore
    Delay Slot: remap %i bank to %o bank;
    pop frame
```

12/31/04

PSU CS322 W05(C) Andrew Tolmach 1997-2005

11

Optimizing Leaf Routines

A routine that does not call any other routines (called a "leaf") can sometimes execute in the **same register window** and stack frame as its caller.

Hence, **no save** and **restore** instructions are used.

Arguments now live in **%o1,%o2**, etc. rather than being shifted into **%i1, %i2**, etc. Similarly, return value goes in **%o0**.

Of course, this only works if routine doesn't trample on any registers that might be used by its caller!

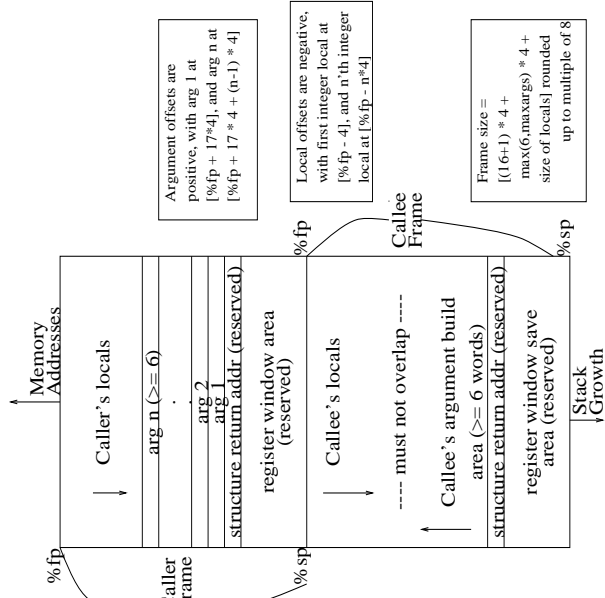
GCC often uses this optimization.

See Architecture Manual Appendix D.5 for details.

12/31/04

PSU CS322 W05(C) Andrew Tolmach 1997-2005

10



12

PSU CS322 W05(C) Andrew Tolmach 1997-2005

12/31/04

Floating Point Registers

There are 32 single precision (32-bit) floating point registers **%f0**, **%f1**, ... completely separate from integer registers, and **not** windowed

By convention, they are managed as “caller-save” registers, i.e. it is the caller's responsibility to save them if it needs to; the callee can use them freely.

Double-precision (64 bit) operations act on **pairs** of adjacent **%f** registers; the first of the pair must have an even number, e.g. **%f2**, **%f3** is a valid pair.

To move a double-precision value from one pair of **%f** registers to another, must do two **fmovs** instructions (**fmovd** is a synthetic instruction).

There is no way to move values directly between integer and fp registers (must use memory!)