Lecture 9 **Sparc Instructions** May 20, 2009

1 Alignment

- We've already discussed alignment in general. For Sparc
- Integers are word aligned (4 bytes)
- Stack is double word aligned (8 bytes) − align all but locals, then align with locals
- To word align `i`: -(-i & -WORD_SIZE)

2 Instruction Set

- Very close to the Intermediate Language discussed previously.
- See handout.

3 Subroutines

- Subroutines will begin with a prologue to setup the stack and to save registers.

```
        .section        ".text"
        .align 4
        .global main
        .type    main, #function
main:
        !#PROLOGUE# 0
        save     %sp, -120, %sp
        !#PROLOGUE# 1
```

  - Set following section as executable text (setting stays until another `.section` pseudo-operation is specified).
  - Align the code properly.
  - Make name accessible externally.
  - Specify the type for the linker.
  - Define the function.
  - The `save` allocates stack space and stores the register window.

- Each subroutine also has an epilogue.

```
ret
restore
.size    main, .-main
```

  - The `ret` and `restore` may differ if optimizing leaf functions.
  - Set size of subroutine (diff of current location and initial def of main).

4 Branch Delay Slot

- Each branch instruction (including `call`, `ret`, and `retl`) has a branch delay slot.
- The instruction in this slot will *always* be executed, whether the branch is taken or not. (Though the Sparc does support nullification, but extra complication).
- Simplest (in most cases) to initially fill this slot with `nop`.
- Fill `ret`'s with `restore`.
- Can fill `call`'s with parameter passing instruction (such as copy to $o0).
- Can fill a branch's with instruction from before or from one of the paths (taking care to respect all register use and other branches).

5 Immediates

- Sparc is a RISC architecture. This implies that the instructions are of a fixed size. In fact, the instructions are 32 bits.
- But, a valid `int` can be 32 bits. So how can such an `int` be used as an immediate in an instruction (say an add)?
- It cannot.
- If immediate takes more than 13 bits ($-4096 \leq$ value ≤ 4095), then must store the 32-bit value in a register.
- But if one cannot use a 32-bit immediate in an instruction, then how can you ever get it into a register?

```
sethi %hi(const32),%reg
or    %reg,%lo(const32),%reg
```

- The assembler does support a synthetic instruction that expands into the two instructions above. `set const32,%reg` (Watch out for delay slots.)

6 Globals

- When generating assembly, we don't necessarily know where a global variable will be placed (and therefore what it's address will be).
- Actual addressing will be taking care of by the linker (and then the loader).
- This is good. It means that the assembly can just use symbols.
- A global can be defined by a `.common` directive (specifying the size and alignment).

```
.common glob_i,4,4
```

- This symbol can then be used as an address (which is 32 bits).

```
sethi   %hi(glob_i), %g1
or      %g1, %lo(glob_i), %o5
```

- Address is now in `%o5`.
- Can store into or load from this address as appropriate.

```
mov     7, %g1
st      %g1, [%o5]
```

7 Strings (printf/scanf)

- Similar to the idea above. Create a label that corresponds to the desired string.

```
    .section        ".rodata"
    .align 8
 .LLC0:
    .asciz  "%d\n"
```

- Then use that label to get the address (32 bits) of the string.

```
sethi   %hi(.LLC0), %g1
or      %g1, %lo(.LLC0), %o0
mov     %g0, %o1
call    printf, 0
 nop
```

- Same idea for scanf, but need address for second argument.

8 sdivx

- There is an poorly-documented issue when using the v9 sdivx instruction.
- The local and input registers cannot be used as operands for this instruction.

Sampling (see website for full instruction set)

Arithmetic

add	r_1, (r_2 | $i_{13\ bits}$), r_3
sub	r_1, (r_2 | $i_{13\ bits}$), r_3

Boolean

and	r_1, (r_2 | $i_{13\ bits}$), r_3
or	r_1, (r_2 | $i_{13\ bits}$), r_3
xor	r_1, (r_2 | $i_{13\ bits}$), r_3

Comparison and Branching

cmp	r_1, (r_2 | $i_{13\ bits}$)	(synthetic instruction)
be	label	
bge	label	
bg	label	
ble	label	
bl	label	
bne	label	
ba	label	

Loads and Stores

ld [address], r_1	(address may be reg + reg, reg ± imm(13), imm(13) + reg, imm(13))
st r_1, [address]	

Invocation

call	label
ret	
save	r_1, (r_2 | $i_{13\ bits}$), r_3
restore	

Miscellaneous

sethi	$i_{22\ bits}$, r_1

```
fib:
loadinargument n, 0, evil.util.ActivationRecord@a12495, r0
loadi 0, r2
comp r0, r2, ccr
cbrne ccr, L3, L2
L2:
loadi 0, r4
storeret r4
jumpi L1
L4:
L3:
loadi 1, r6
comp r0, r6, ccr
cbrgt ccr, L8, L7
L7:
loadi 1, r8
storeret r8
jumpi L5
L8:
loadi 1, r10
sub r0, r10, r11
storeoutargument r11, 0
call fib, 1
loadret r12
loadi 2, r14
sub r0, r14, r15
storeoutargument r15, 0
call fib, 1
loadret r16
add r12, r16, r17
storeret r17
L1:
L5:
L6:
L11:
ret true

main:
addi rarp, val, r1
read r1
loadai rarp, val, r0
loadi 0, r3
comp r0, r3, ccr
cbrge ccr, L14, L13
L13:
loadi -1, r5
```

```
storeret r5
jumpi L12
L15:
L14:
storeoutargument r0, 0
call fib, 1
loadret r7
println r7
loadi 0, r8
storeret r8
L12:
L16:
L17:
ret true
```

```
.section        ".text"
.align 4
.global fib
.type    fib, #function
fib:
!#PROLOGUE# 0
save     %sp, -112, %sp
!#PROLOGUE# 1
mov      %i0, %i0
mov      %i0, %o1
mov      %g0, %o0
cmp      %o1, %o0
bne      .L1
nop
.L2:
mov      %g0, %i0
mov      %i0, %i0
ba       .L3
nop
.L4:
.L1:
mov      %i0, %o1
mov      1, %o0
cmp      %o1, %o0
bg       .L5
nop
.L6:
mov      1, %i0
mov      %i0, %i0
ba       .L7
nop
.L5:
mov      %i0, %o1
mov      1, %o0
sub      %o1, %o0, %o0
mov      %o0, %o0
call     fib
nop
mov      %o0, %i1
mov      %i0, %o1
mov      2, %o0
sub      %o1, %o0, %o0
mov      %o0, %o0
call     fib
nop
mov      %o0, %o0
```

```
add     %i1, %o0, %i0
mov     %i0, %i0
.L3:
.L7:
.L8:
.L9:
ret
restore
.size    fib, .-fib
.align 4
.global main
.type    main, #function
main:
!#PROLOGUE# 0
save     %sp, -120, %sp
!#PROLOGUE# 1
add     %fp, -20, %o1
sethi   %hi(.LLC2), %o0
or      %o0, %lo(.LLC2), %o0
mov     %o1, %o1
call    scanf
nop
ldsw    [%fp-20], %o1
mov     %o1, %o1
mov     %g0, %o0
cmp     %o1, %o0
bge     .L10
nop
.L11:
mov     -1, %i0
mov     %i0, %i0
ba      .L12
nop
.L13:
.L10:
mov     %o1, %o1
mov     %o1, %o0
call    fib
nop
mov     %o0, %o1
sethi   %hi(.LLC1), %o0
or      %o0, %lo(.LLC1), %o0
mov     %o1, %o1
call    printf
nop
mov     %g0, %i0
mov     %i0, %i0
.L12:
.L14:
.L15:
ret
restore
.size    main, .-main

.section        ".rodata"
.align 8
.LLC0:
.asciz  "%d "
.align 8
.LLC1:
.asciz  "%d\n"
.align 8
.LLC2:
.asciz  "%d"
```