Lecture 6

or “Everything is Awesome”
Announcements

Homework due at the beginning of next class! I will send out an email reminding everyone.
More Announcements

I hate the Seahawks
Finish stuff from last time
Bitwise operations (and ex)

AND: and, andi

and $t0, $t1, $t2  # $t0 = $t1 & $t2
andi $t0, $t1, 4   # $t0 = $t1 & 4

OR: or, ori

or $t0, $t1, $t2   # $t0 = $t1 | $t2
ori $t0, $t1, 4   # $t0 = $t1 | 4
More Bitwise

NOT: nor

```
nor $t0, $t1, $t2      # $t0 = ~(t1 | t2)
```

What can we do to NOT $t1 and store it in $t0?
More Bitwise

NOT: nor

nor $t0$, $t1$, $t2$  #: $t0 = \neg(t1 \mid t2)$

What can we do to NOT $t1$ and store it in $t0$?
$t0 = \neg(t1 \mid t1)$
Shifts

Shift left logical: sll

\[ \text{sll } \$t2, \$s0, 4 \quad \# \quad \$t2 = \$s0 \ll 4 \text{ bits} \]

\[ \begin{align*}
00001101 \text{ by } 4 & = 11010000 \\
00001101 & = 13 \\
11010000 & = 208 \\
208 & = 13 \times 2^4 \quad (4 \text{ comes from shift amount})
\end{align*} \]
Shifts

Shift right logical: srl

srl $t2, $s0, 4  # $t2 = $s0 >> 4 bits

00101001 by 4 = 00000010

00101001 = 41

00000010 = 2

2 = 41/2^4 (rounded down) (41/16 = 2.5625)
Shifts

Shift right-arithmetic: SRA (keep the sign)

\[
sra \ \$t2, \ \$s0, \ 2 \ # \ \$t2 = \$s0 >> 2 \ \text{bits}
\]

\[
11110101 \ \text{by} \ 2 = 11111101
\]

\[
11110101 = -11
\]

\[
11111101 = -3
\]

\[-11/2^2 = -3\]
More shifts

SLLV, SRLV, SRAV

These are for if the shift amount is in a register
We will use branching to implement if statements, loops, and procedure calls
Branching

2 types of branches, unconditional and conditional

Unconditional: j <label> # jump to this spot
Conditional: beq, bgez, bgezal, bgtz, blez

Branch targets are 16 bit immediate offset
C-code:

```
if(i==j)  f = g+h;
else f = g-h;
```

f, g, h, i, and j are in $s0-$s4 respectively.
if(i==j)          # we will do this line by line
    f = g+h;
else
    f = g-h;
if(i==j) # we will do this line by line
    f = g+h;
else
    f = g-h;
First one:
  
  beq $s3, $s4, L1
  sub $s0, $s1, $s2
  j Exit

L1:  add $s0, $s1, $s2

Exit:  ...
First one:

```
bne $s3, $s4, L1
add $s0, $s1, $s2
j Exit

L1:  sub $s0, $s1, $s2
```

Exit: ...
Loops

C code:

```
while (save[i] == k) i += 1;
```

i in $s3$, k in $s5$, save starts $s6$
while (save[i] == k) i += 1;

Loop:
- `sll $t1, $s3, 2`
- `add $t1, $t1, $s6`
- `lw $t0, 0($t1)`
- `bne $t0, $s5, Exit`
- `addi $s3, $s3, 1`

Exit: ...

Loops MIPS
More Conditional Ops

```
slt rd, rs, rt    # set on less than
    if(rs < rt) rd = 1; else rd = 0;
slti rt, rs, constant
    if(rs < constant) rt = 1; else rt = 0;

Use in combination with beq, bne
    slt $t0, $s1, $s2        # if ($s1 < $s2)
    bne $t0, $zero, L      #branch to L
```
Next time

Assembly Language Paradigm