Lecture 9

On branching -- how can we implement conditional logic and loops in assembly language? How do we translate C++ programs using those elements to MIPS?

Branching and Loops

- To implement if statements in MIPS, we can use "branching"
- All branch instructions do comparisons
- BEQ (branch if equal), BNE (branch if not equal)
  - `BEQ $s0, $s1, LABEL1` *(if $s0 == $s1, jump to LABEL1)*
- A label in a programming language is a sequence of characters that identifies a location within source code

Example

C++

```
if (i != 0) i++;
```

MIPS *Let i be in $s0*

```
BEQ $s0, $zero, L1
ADDI $s0, $s0, 1
L1:
```

This works because we jump over the ADDI instruction if we meet the condition

- We can also use branching to implement loops

C++

```
for(int i = 0; i < 10; ++i)
    var++;
```

MIPS

```
    ADDI $t0, $zero, $zero      #initialize start, i = 0
    ADDI $t1, $zero, 10         #initialize stop, i = 10
LOOP:   BEQ $t0, $t1, END       #if i == 10, end the loop
        ADDI $s0, $zero, 1      # var++, assuming var is in s0
        ADDI $t0, $zero, 1      # increment counter, i++
        BEQ $zero, $zero, LOOP  # always loop back to the top
END:
```

- The basic idea is that we keep looping until our index variable and the stop are equal, at which point we jump over the loop to the following instructions.
- Notice the line `BEQ $zero, $zero, LOOP` -- this always evaluates to true, so we loop back every time we finish executing the body of the loop. There's actually a specific instruction that is meant to jump unconditionally.
- `J` -- unconditional jump
  - *In our example, we would replace that line with* `J LOOP`

Comparing values (less than)

- What would happen if our loop above incremented i by 3 instead of by 1?
- Our trick of checking if i == 10 to determine when to stop would fail, because i would skip over 10 (straight to 12), and the loop would continue indefinitely, since the condition is never met.
- A solution would be to use the *less than* operator!

SLT -- Set on less than (signed)

- Compares two registers, if the value in the first is less than the value in the second, it "sets" the first bit of the destination register (makes the rightmost bit 1)
- `SLT $s0, $t0, $t1` *# is t0 < t1? set s0 if yes*
- Of course we also have `SLTI` , which does the same thing but accepts an immediate