CS 237 Meeting 6 — 9/19/12

Announcements

1. Homework 1 is due Friday.

DIV and MULT

1. You were forced to learn DIV in lab, so I just want to give you a quick review and chance to ask questions.

More on Branches and Conditionals

1. Last time I quickly showed you code for the main loop in my integer square root program.
2. Most of the code in the loop implements printfs. We are experts at this now.
3. The interesting part of the code is the code that implements the loop. This takes a simple and standard form. Given a while loop of the form

```
while ( condition ) {
statements
}
```

   the corresponding assembly language code will always look like:

```
loopStartLabel:
 assembly code for condition leaving result (0 or 1) in $r
beq  $r,$0,loopDoneLabel

 assembly code for the statements in the loop body

j  loopStartLabel

loopDoneLabel:
 code for rest of function
```

4. We can apply this to the task of writing the loop for dumbSqrt.
5. First, we need to discover the slt instruction
6. Similar templates can be used for if statements and for loops

Implementing simple functions

1. We can also take a template approach to implementing simple functions
2. At the top of the function, we put a label for the name of the function and a comment assigning $an registers to its parameters
3. Then, we translate its code with the extra rule that whenever we see a return statement we generate code to move the value of the return expression to $v0 and execute "jr $ra".

Other instructions we never got to — shifts and logicals

1. Shifts can be used for multiplication and division by powers of 2 but:
   - Be careful to use the arithmetic shift right rather than logical shift right when doing division of signed numbers because it fills the sign bits correctly.
   - Be aware that if you use shifts to divide, -5/2 = -3 even though 5/3 = 2.
2. Do not use the logical instructions MIPS provides to implement the C && and || operators. These operators are expected to only evaluate enough of the expressions that provide their operand values to enable them to determine the result. For example, if the first operand of a && returns false, the whole expression can return false without ever evaluating the expression for the second operand. To do this, use branches to implement && and ||.