Notes for 8/31/09

\[ f = (g + h) - (i + j) \]
\[ f, g, h, i, j := s0, s1, s2, s3, s4 \]

How could we write this in MIPS assembly?

We will write this as though \( f \ldots j \) are global variables, not stack variables.

\[
\begin{align*}
\text{add } &\$t0, \$s1, \$s2 \\
\text{add } &\$t1, \$s3, \$s4 \\
\text{sub } &\$s0, \$t0, \$t1 \\
\text{la } &\$t7, f \\
\text{sw } &\$s0, 0(\$t7)
\end{align*}
\]

\[ A := s3 \]
\[ h := s2 \]

How could we write this in MIPS assembly?

To allocate an int array of size 20 (assume ints are 32 bit) as though it were a global variable, we could write this line in the .data section:

\[ A: \text{.space 80} \]

Then in our .text section...

\[
\begin{align*}
\text{lw } &\$t0, 32(\$s3) \\
\text{add } &\$t0, \$s2, \$t0 \\
\text{sw } &\$t0, 48(\$s3)
\end{align*}
\]

\[ A[i] \]
\[ A := s3 \]
\[ i := s4 \]

How could we write this in MIPS assembly?

\[
\begin{align*}
\text{sll } &\$t6, \$s4, 2 \quad \text{// multiply } \$s4 \text{ by 4, size of a 32-bit int} \\
&\quad \text{// logical shifts are faster than multiplication} \\
&\quad \text{// this optimization is called strength reduction} \\
\text{add } &\$t7, \$t6, \$s3 \\
\text{lw } &\$t8, 0(\$t7)
\end{align*}
\]
Branches in MIPS

if (a != b) {
    ...
} else {
    ...
}
a, b := $t0, $t1

How could we write this in MIPS assembly?

beq $t0, $t1, LabelA // LabelA is 16 bits, so we can only jump
    // +/-32768 from the program counter
    ...
    b End // jump unconditionally to End
LabelA:
    ...
End:

if (a == b) {
    ...
} else {
    ...
}
a, b := $t0, $t1

How could we write this in MIPS assembly?

beq $t0, $t1, LabelA // branch if not equal this time
    ...
    b End // jump unconditionally to End
LabelA:
    ...
End:

if (a < b) {
    ...
} else {
    ...
}
a, b := $t0, $t1

How could we write this in MIPS assembly?
Without using a branch pseudo-instruction:

```
slt $t7, $t0, $t1      // set $t7 to 1 if $t0 < $t1
bne $t7, $zero, LabelA // branch if $t7 is not zero
      . . .
b End                  // jump unconditionally to End
LabelA:
      . . .
End:
```

Using a branch pseudo-instruction:

```
blt $t0, $t1, LabelA // branch if $t0 < $t1
      . . .
b End                  // jump unconditionally to End
LabelA:
      . . .
End:
```

if (a <= b) {
  ...
} else {
  ...
}

```
a, b := $t0, $t1
```

How could we write this in MIPS assembly?

Without using a branch pseudo-instruction:

```
slt $t7, $t1, $t0      // set $t7 to 1 if $t1 < $t0
beq $t7, $zero, LabelA // branch if $t7 is not zero
      . . .
b End                  // jump unconditionally to End
LabelA:
      . . .
End:
```

Using a branch pseudo-instruction:

```
bgt $t1, $t0, LabelA // branch if $t1 > $t0
      . . .
b End                  // jump unconditionally to End
LabelA:
      . . .
End:
```