CSCI 210: Computer Organization
Lecture 10: Control Flow

Stephen Checkoway

Slides from Cynthia Taylor

CS History: The If-Else Statement

- Haskell Curry and Willa Wyatt are the first people to describe performing different instructions based on the result of a previous calculation, on the Eniac in 1946
- Early assembly language instructions jumped to a new memory location based on a specific condition, were not general purpose
- Fortran (1957) specifying jumps to three locations at once, depending on whether a calculation was negative, zero, or positive, and gave it the name "if.”
- Flow-matic (Grace Hopper, 1958), used comparisons between numbers and used the name “otherwise” for else
- In 1958, a German computing organization proposed an if statement that took an arbitrary Boolean statement, had an ”else” case, and returned control to immediately after the if/else statement after completing the statement

Today: Program control flow

- High level languages have many ways to control the order of execution in a program: if, if-else, for loops, while loops

- Today we will look at how these higher order concepts are built out of MIPS control flow instructions

Control Flow

- Recall the basic instruction cycle
  - IR = Memory[PC]
  - PC = PC + 4
- Both branch and jump instructions change the value of the program counter

Control Flow - Instructions

- Conditional
  - beq, bne: compare two registers and branch depending on the comparison
  - Change the value of the program counter if a condition is true

- Unconditional
  - j, jal, jr: jump to a location
  - Always change the value of the program counter

Control Flow - Labels

- In assembly, we use labels to help us guide control flow. Labels can be the target of branch or jump instructions.
- Example:

```
j Label
…
Label:  add $t1, $t0, $t2
```

- Assemblers are responsible for translating labels into addresses.

C Code

```
if (X == 0)
  X = Y + Z;
```

Assuming X, Y, and Z are integers in registers $t0, $t1, and $t2, respectively, which are the equivalent assembly instructions?

A

```
       beq $t0,$zero, Label
Label: add $t0, $t1, $t2
```

B

```
       beq $t0,$zero, Label
       add $t0, $t1, $t2
Label:
```

C

```
       bne $t0,$zero, Label
       add $t0, $t1, $t2
Label:
```

D – None of these is correct.

If (x < y): Set Less Than

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- `slt rd, rs, rt`
  - 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
```

Branch Instruction Design

- Why not `blt`, `bge`, etc?
- Hardware for <, ≥, … slower than =, ≠
  - Combining with branch involves more work per instruction
  - `beq` and `bne` are the common case

High level code often has code like this:

```
if (i < j) {
    i = i + 1;
}
```

Assume \$t0 holds *i* and \$t1 holds *j*. Which of the following is the correct translation of the above code to MIPS assembly (recall \$zero is always 0):

A

```
    slt   $t2, $t0, $t1
    bne   $t2, $zero, x
    addi  $t0, $t0, 1
x:  next instruction
```

B

```
    slt   $t2, $t0, $t1
    bne   $t2, $zero, x
x:  addi  $t0, $t0, 1
    next instruction
```

C

```
    slt   $t2, $t0, $t1
    beq   $t2, $zero, x
    addi  $t0, $t0, 1
x:  next instruction
```

D None of the above

```
slt rd, rs, rt
```

if (rs < rt) rd = 1; else rd = 0;

Signed vs. Unsigned

- Signed comparison: `slt`, `slti`
- Unsigned comparison: `sltu`, `sltui`

slt vs sltu

\$s0 = 1111 1111 1111 1111 1111 1111 1111 1111

\$s1 = 0000 0000 0000 0000 0000 0000 0000 0001

	slt \$t0, \$s0, \$s1	**sltu \$t0, \$s0, \$s1**
A	\$t0 = 1	\$t0 = 1
B	\$t0 = 0	\$t0 = 1
C	\$t0 = 0	\$t0 = 0
D	\$t0 = 1	\$t0 = 0

```
slt rd, rs, rt
```

if (rs < rt) rd = 1; else rd = 0;

Questions on BEQ, BNE, SLT?

Reading

- Next lecture: Procedures
  - Section 2.9
- Problem set: Due Friday
- Lab 2: Due Monday