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.

Jump

- `j label`
  - Go directly to the label (i.e., PC = label)

- `jal label`
  - Go directly to the label (i.e., PC = label) and set the link register (we'll discuss this later)

- `jr register`
  - Go directly to the address specified in the register

High-level 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 then rd = 1 else rd = 0
- `slti rt, rs, constant`
  - if rs < constant then 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
  - MIPS philosophy: fewer, simpler instructions

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?

How to access an array in a for loop

- Can’t programmatically change the offset

- Need to change the *base address* instead

- Add 4 to the base address every time you want to move to the next element of the array (assuming an array of 4-byte values)

```
for (i=0; i < 10; i++){
  A[i] = 0;
}
```

*Assume base address of A is in $s0

```
      move  $t0, $zero
      li    $t1, 40
loop: beq   $t0, $t1, end
      add   $t2, $s0, $t0
      sw    $zero, 0($t2)
      addi  $t0, $t0, 4
      j     loop
end:
```

C Code

```
for (i = 0; i < 10; i++){
   A[i+1] = A[i];
}
```

Assume the base address of A is in $t0, and i is in $t1. Each element of A is 4 bytes. What is the equivalent assembly?

```
      addi $t2, $zero, 10
      add  $t1, $zero, $zero
for:  bne  $t1, $t2, end
      lw   $t3, $t1($t0)
      addi $t1, $t1, 1
      sw   $t3, $t1($t0)
      j    for
end:
```

A

```
      addi $t2, $zero, 40
      add  $t1, $zero, $zero
for:  beq  $t1, $t2, end
      add  $t4, $t0, $t1
      lw   $t3, 0($t4)
      addi $t1, $t1, 4
      add  $t4, $t0, $t1
      sw   $t3, 0($t4)
      j    for
end:
```

B

```
      addi $t2, $zero, 10
      add  $t1, $zero, $zero
      bne  $t1, $t2, end
      add  $t4, $t0, $t1
      lw   $t3, 0($t4)
      addi $t1, $t1, 1
      add  $t4, $t0, $t1
      sw   $t3, 0($t4)
end:
```

C

D – More than one of these

E – None of these

C Code

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

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

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

A

```
        bne $t0, $zero, false
        add $t0, $t1, $t2
        j endif
false:  add $t0, $t2, $t2
endif:
```

B

```
        bne $t0, $zero, false
        j endif
        add $t0, $t1, $t2
false:  add $t0, $t2, $t2
endif:
```

C

D – None of the above

C Code

```
while (i < 10){
    i = i + 1;
}
```

Assume i is in \$t0. What is the equivalent assembly?

```
slti rt, rs, imm
```

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

```
w:  slti $t2, $t0, 10
    beq $t2, $zero, end
    addi $t0, $t0, 1
    j w
end:
```

A

```
w:  slti $t2, $t0, 10
    beq $t2, $zero, end
    addi $t0, $t0, 1
end:
```

B

```
    slti $t2, $t0, 10
w:  beq $t2, $zero, end
    addi $t0, $t0, 1
    j w
end:
```

C

D – More than one of these

E – None of these