

CSCI 210: Computer Organization

Lecture 11: Control Flow

Stephen Checkoway

Oberlin College

Mar. 14, 2022

Slides from Cynthia Taylor

Announcements

- Problem Set due Friday
- Lab 2 due Sunday
- Office Hours Tuesday 13:30 – 14:30

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
 - $IR = \text{Memory}[PC]$
 - $PC = PC + 4$
- 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 ($rs < \text{constant}$) $rt = 1$; 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 has *i* and \$t1 has *j*. Which of the following is the correct translation of the above code to MIPS assembly (recall \$zero is always 0):

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

A

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

B

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

C

D

None of the above

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

Signed vs. Unsigned

- Signed comparison: `slt`, `sltui`
- 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;

Jump! Jump!

- `j label`
 - Go directly to the label (i.e., $PC = \text{label}$)
- `jr register`
 - Go directly to the address specified in the register

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?

A

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

B

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

C

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

D – None of the above

C Code

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

Assume the base address of A is in \$t0 and sum is in \$s0.

Elements of A are words. What is the equivalent assembly?

A

```
li    $t2, 10  
move  $t1, $zero  
for:beq $t1, $t2, end  
lw    $t3, $t1($t0)  
add   $s0, $s0, $t3  
addi  $t1, $t1, 1  
j     for  
end:
```

B

```
li    $t2, 10  
move  $t1, $zero  
for:beq $t1, $t2, end  
lw    $t3, 0($t0)  
add   $s0, $s0, $t3  
addi  $t0, $t0, 4  
addi  $t1, $t1, 1  
j     for  
end:
```

C – More than one of these

D – None of these

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 up an element of the array


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

*Assume base address of A is in \$s3

```
        move    $s0, $zero  
        li      $s1, 40  
loop:    beq     $s0, $s1, end  
        add     $s4, $s3, $s0  
        sw      $zero, 0($s4)  
        addi    $s0, $s0, 4  
        j       loop  
end:
```

Jump and Link

`jal Label`

- Address of following instruction put in \$ra
- Jumps to target address

What is the most common use of a jal instruction and why?

	Most common use	Best answer
A	Procedure call	Jal stores the next instruction in your current function so the called function knows where to return to.
B	Procedure call	Jal enables a long jump and most procedures are a fairly long distance away
C	If/else	Jal lets you go to the if while storing pc+4 (else)
D	If/else	Jal enables a long branch and most if statements are a fairly long distance away
E	None of the above	

Reading

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