
IC220
SlideSet #3: Control Flow
(Section 2.7, plus pgs 86-87,
128-129,140-141)

1

Example

- What is the MIPS assembly code for the following:

```
    if (i == j)
        go to L1;
    f = g + h;
L1:  f = f - i;
```

Variables f to j are assigned to registers \$s0 to \$s4

```
f $s0
g $s1
h $s2
i $s3
j $s4
```

3

Conditional Control

- Decision making instructions
 - alter the control flow,
 - i.e., change the "next" instruction to be executed

- MIPS *conditional* branch instructions (I-type):

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- Example: if (i == j)
 h = i + j;

- Assembly Code:

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label:  ....
```

2

Unconditional Control

- MIPS unconditional branch instructions:

```
    j label
```

- New type of instruction (J-type)
 - op code is 2 (no function field)

- Example:

```
if (i!=j)                beq $s4, $s5, Lab1
    h=i+j;                add $s3, $s4, $s5
else                       j Lab2
    h=i-j;                Lab1: sub $s3, $s4, $s5
                           Lab2: ...
```

4

Example

- What is the MIPS assembly code for the following:

if (i == j) f = g + h;	f \$s0
else f = g - h;	g \$s1
	h \$s2
	i \$s3
	j \$s4
- Variables f to j are assigned to registers \$s0 to \$s4

5

Control Flow – Branch if less than

- We have: beq, bne, what about Branch-if-less-than?
- New instruction:

	if \$s1 < \$s2 then
slt \$t0, \$s1, \$s2	\$t0 = 1
	else
	\$t0 = 0
- slt is a R-type instruction (function code 42)

7

So far:

Ex 2-11 to 2-13

- Instruction** **Meaning**

add \$s1, \$s2, \$s3	\$s1 = \$s2 + \$s3
sub \$s1, \$s2, \$s3	\$s1 = \$s2 - \$s3
lw \$s1, 100(\$s2)	\$s1 = Memory[\$s2+100]
sw \$s1, 100(\$s2)	Memory[\$s2+100] = \$s1
bne \$s4, \$s5, L	Next instr. is at Label if \$s4 != \$s5
beq \$s4, \$s5, L	Next instr. is at Label if \$s4 == \$s5
j Label	Next instr. is at Label

- Formats:**

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

6

Example

- What is the MIPS assembly code to test if variable a (\$s0) is less than variable b(\$s1) and then branch to Less: if the condition holds?

	if (a < b)
	go to Less;

Less:

8

Pseudoinstructions

- Example #1: Use `slt` instruction to build "`b1t $s1, $s2, Label`"
 - “Pseudoinstruction” that assembler expands into several real instructions
 - Note that the assembler needs a register to do this
 - What register should it use?
- Why not make `b1t` a real instruction?
- Example #2: “Move” instruction
 - “`move $t0, $t1`”
 - Implementation?

9

Constants

- Small constants are used quite frequently
 - e.g., `A = A + 5;`
 - `B = B + 1;`
 - `C = C - 18;`
- Possible solution
 - put ‘typical constants’ in memory and load them.
 - And create hard-wired registers for constants like zero, one.
- Problem?
- MIPS Instructions:


```
addi $29, $29, 4
slti $8, $18, 10
andi $29, $29, 6
ori $29, $29, 4
```
- How do we make this work?

I-type	op	rs	rt	
--------	----	----	----	--

11

Policy of Use Conventions

Name	Register number	Usage
<code>\$zero</code>	0	the constant value 0
<code>\$v0-\$v1</code>	2-3	values for results and expression evaluation
<code>\$a0-\$a3</code>	4-7	arguments
<code>\$t0-\$t7</code>	8-15	temporaries
<code>\$s0-\$s7</code>	16-23	saved
<code>\$t8-\$t9</code>	24-25	more temporaries
<code>\$gp</code>	28	global pointer
<code>\$sp</code>	29	stack pointer
<code>\$fp</code>	30	frame pointer
<code>\$ra</code>	31	return address

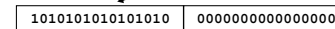
`$at` = Register #1 – reserved for assembler
`$k0, $k1` = Register #26, 27 – reserved for OS

10

How about larger constants?

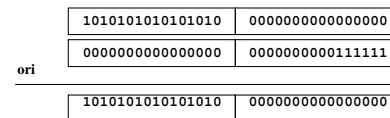
- We'd like to be able to load a 32 bit constant into a register
- Must use two instructions, new “load upper immediate” instruction

```
lui $t0, 1010101010101010
```



- Then must get the lower order bits right, i.e.,

```
ori $t0, $t0, 0000000000111111
```



12

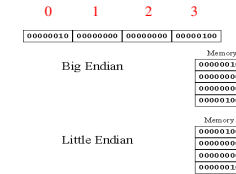
Assembly Language vs. Machine Language

- Assembly provides convenient symbolic representation
 - much easier than writing down numbers
 - e.g., destination first
- Machine language is the underlying reality
 - e.g., destination is no longer first
- Assembly can provide 'pseudoinstructions'
- When considering performance you should count

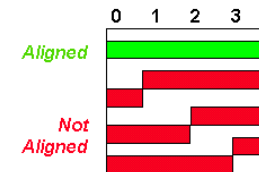
13

Memory – Byte Order & Alignment

- Endian
 - Processors don't care
 - Big: 0,1,2,3
 - Little: 3,2,1,0
 - Network byte order:



- Alignment
 - require that objects fall on address that is multiple of
 - Legal word addresses:
 - Legal byte addresses:



14

Looping

- We know how to make decisions, but:
 - Can we set up a flow that allows for multiple iterations?
 - What high level repetition structures could we use?
 - What MIPS instructions could we use?
- “Basic block”
 - Sequence of instructions _____ except possibly _____ -

15

Looping Example

Ex 2-21 to 2-23

```

Goal: Provide the comments # to the assembly language
C Code
do {
    g = g + A[i];           //vars g to j in $s1 to $s4
    i = i + j;             // $s5 holds base add of A
} while (i != h)

Assembly Language
Loop:  add $t1, $s3, $s3    #
      add $t1, $t1, $t1    #
      add $t1, $t1, $s5    #
      lw $t0, 0($t1)      #
      add $s1, $s1, $t0    #
      add $s3, $s3, $s4    #
      bne $s3, $s2, Loop  #
    
```

16