Testing and Branching

CSE 410, Spring 2007
Computer Systems

http://www.cs.washington.edu/410

Reading and References

• Computer Organization and Design
  » Section 2.5, Logical Operations
  » Section 2.6, Instructions for Making Decisions
  » (CD) Section A.9, SPIM
  » (CD) Section A.10 through page A-50, MIPS R2000
  Assembly Language

goto considered harmful

• “Oh what a tangled web we weave, When first we practice to deceive!”
  » Sir Walter Scott
• Branching in assembly language can turn your program into a rat’s nest that cannot be debugged
• Keep control flow simple and logical
• Use comments describing the overall logic
  » (if, while, for, … pseudo-code is often great!)

Conditional Branch

A change in the program’s flow of control that depends on some condition

yes

no

...
Branch instructions

- Branch instructions are I-format instructions
  » op code field
  » two register fields
  » 16-bit offset field
- Simplest branches check for equality
  » `beq $t0, $t1, address`
  » `bne $t0, $t1, address`

```
if (i==j) then a=b;
```

- Assume all values are in registers
- Note that the test is inverted compared to `if`!

```
# t0=i, t1=j, s0=a, s1=b
bne $t0, $t1, skip
move $s0, $s1
```

```
while (s[i]==k) i = i+j;
```

```
# s0=addr(s), v1=i, a0=k, a1=j
loop:
sll $v0,$v1,2  # v0 = 4*i
addu $v0,$s0,$v0  # v0 = addr(s[i])
lw $v0,0($v0)  # v0 = s[i]
addu $v1,$v1,$a1 # i = i+j
beq $v0,$a0,loop # loop if equal
subu $v1,$v1,$a1 # i = i-j
```

```
for (i=0; i<10; i++) s[i] = i;
```

```
# s0=addr(s), t1=i
move $t1,$zero  # i = 0
loop:
sll $t0,$t1,2  # t0 = i*4
addu $t0,$s0,$t0  # t0 = addr(s[i])
sw $t1,0($t0)  # s[i] = i
addu $t1,$t1,1  # i++
slt $t0,$t1,10  # if (i<10) t0=1
bnez $t0,loop  # loop if (i<10)
```
How do we encode the destination?

- Calculating the destination address
  - \( 4 \times (\text{the 16-bit offset value}) \)
  - is added to the Program Counter (PC)
- The offset is a word offset in this case
- The base register is always the PC, so we don’t need to specify it in the instruction
- Covers a range of \( 2^{16} \) words (64 KW)

Comparison instructions

- For comparisons other than equality
  - \( \text{slt} \) : set less than
  - \( \text{sltu} \) : set less than unsigned
  - \( \text{slti} \) : set less than constant value
  - \( \text{sltiu} \) : set less than unsigned constant
- set \( t0 \) to 1 if \( t1 < t2 \)
  \[ \text{slt} \ t0, \ t1, \ t2 \]

Pseudo-instructions

- The assembler is your friend and will build instruction sequences for you
- Original code:
  \[
  \text{bge} \ \$a0, \$t1, \text{end} \quad \# \text{ if } a0 \geq t1 \text{ skip}
  \]
- Actual instructions:
  \[
  \text{slt} \ \$at, \$a0, \$t1 \quad \# \text{ if } a0 < t1 \text{ at=true}
  \text{beq} \ \$at, \$0, \text{end} \quad \# \text{ skip if } at==\text{false}
  \]

Jump Instructions

- Jump instructions provide longer range than branch instructions
- 26-bit word offset in J-format instructions
  - \( \text{j} \) : jump
  - \( \text{jal} \) : jump and link (store return address)
- 32-bit address in register jumps
  - \( \text{jr} \) : jump through register
  - \( \text{jalr} \) : jump through register and link
J-format fields

- The word offset value is multiplied by 4 to create a byte offset
  » the result is 28 bits wide
- Then concatenated with top 4 bits of PC to make a 32 bit destination address

Important Jumps

- Jump and link (jal)
  » call procedure and store return address in $ra
- Jump through register (jr)
  » return to caller using the address in $ra
- We will talk about procedure calls in excruciating detail shortly