Flow of Control -- Conditional branch instructions

- You can compare directly
  - Equality or inequality of two registers
  - One register with 0 (>, <, ≥, ≤)

- and branch to a target specified as
  - a signed displacement expressed in *number of instructions* (not number of bytes) from the instruction *following* the branch
  - in assembly language, it is *highly* recommended to use labels and branch to labeled target addresses because:
    - the computation above is too complicated
    - some pseudo-instructions are translated into two real instructions
Examples of branch instructions

beq rs,rt,target #go to target if rs = rt
beqz rs, target #go to target if rs = 0
bne rs,rt,target #go to target if rs != rt
bltz rs, target #go to target if rs < 0
etc.

but note that you cannot compare directly 2 registers for <, > …
Any idea why?
Comparisons between two registers

• Use an instruction to set a third register
  
  \[
  \text{slt} \quad \text{rd}, \text{rs}, \text{rt} \quad \# \text{rd} = 1 \text{ if } \text{rs} < \text{rt} \text{ else rd} = 0
  \]
  
  \[
  \text{sltu} \quad \text{rd}, \text{rs}, \text{rt} \quad \# \text{same but rs and rt are considered unsigned}
  \]

• Example: Branch to Lab1 if $5 < $6
  
  \[
  \text{slt} \quad \$10, \$5, \$6 \quad \# \$10 = 1 \text{ if } \$5 < \$6 \text{ otherwise } \$10 = 0
  \]
  
  \[
  \text{bnez} \quad \$10, \text{Lab1} \quad \# \text{branch if } \$10 = 1, \text{ i.e., } \$5 < \$6
  \]

• There exist pseudo instructions to help you!
  
  \[
  \text{blt} \quad \$5, \$6, \text{Lab1} \quad \# \text{pseudo instruction translated into}
  \]
  
  \[
  \quad \# \text{slt} \quad \$1, \$5, \$6
  \]
  
  \[
  \quad \# \text{bne} \quad \$1, \$0, \text{Lab1}
  \]

Note the use of register 1 by the assembler and the fact that computing the address of Lab1 requires knowledge of how pseudo-instructions are expanded.
Unconditional transfer of control

- Can use “beqz $0, target”
  - Very useful but limited range (± 32K instructions)
- Use of Jump instructions
  
  \[
  \begin{align*}
  j & \quad \text{target} \quad \# \text{special format for target byte address (26 bits)} \\
  jr & \quad \$rs \quad \# \text{jump to address stored in rs (good for switch statements and transfer tables)}
  \end{align*}
  \]

- Call/return functions and procedures

  \[
  \begin{align*}
  \text{jal} & \quad \text{target} \quad \# \text{jump to target address; save PC of following instruction in $31 (aka $ra)} \\
  jr & \quad \$31 \quad \# \text{jump to address stored in $31 (or $ra)}
  \end{align*}
  \]

  Also possible to use \( \text{jalr rs,rd} \) \quad \# \text{jump to address stored in rs; rd = PC of following instruction in rd with default rd = $31}
Branch addressing format

- Need Opcode, one or two registers, and an offset
  - No base register since offset added to PC
- When using one register (i.e., compare to 0), can use the second register field to expand the opcode
  - similar to function field for arith instructions

```
beq  $4,$5,1000
bgtz $4,1000
```

```
Opc  rs  rt/func  target offset

beq  $4,$5,1000
bgtz $4,1000
```
Example -- High-level language

```c
int a[100];
int i;

for (i=0; i<100; i++){
    a[i] = 5;
}
```
Assembly language version

Assume: start address of array a in r15.
We use r8 to store the value of i and r9 for the value 5

    add       $8,$0,$0        #initialize i
    li           $9,5               #r9 has the constant 5

Loop:  mul       $10,$8,4        #r10 has i in bytes
          #could use a shift left by 2
    addu      $14,$10,$15  #address of a[i]
    sw         $9,0($14)       #store 5 in a[i]
    addiu     $8,$8,1          #increment i
    blt          $8,100,Loop  #branch if loop not finished
          #taking lots of liberty here!
Machine language version (generated by SPIM)

[0x00400020] 0x00004020  add $8, $0, $0 ; 1: add $8,$0,$0
[0x00400024] 0x34090005  ori $9, $0, 5 ; 2: li $9,5
[0x00400028] 0x34010004  ori $1, $0, 4 ; 3: mul $10,$8,4
[0x0040002c] 0x01010018  mult $8, $1
[0x00400030] 0x00005012  mflo $10
[0x00400034] 0x014f7021  addu $14, $10, $15 ; 4: addu $14,$10,$15
[0x00400038] 0xadc90000  sw $9, 0($14) ; 5: sw $9,0($14)
[0x0040003c] 0x25080001  addiu $8, $8, 1 ; 6: addiu $8,$8,1
[0x00400040] 0x29010064  slti $1, $8, 100 ; 7: blt $8,100,Loop
[0x00400044] 0x1420fff9  bne $1, $0, -28 [Loop-0x00400044]