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
Bitz rs, target # go to target if rs < 0
etc.

but note that you cannot compare directly 2 registers for <, > ...

Comparisons between two registers

- Use an instruction to set a third register
  slt rd, x, rt # rd = 1 if x < rt else rd = 0
  sllx rd, x, rt # frame but x and rt are considered unsigned

Example: Branch to Lab1 if $5 < $6
  slt $10, $5, $6 # $10 = 1 if $5 < $6; otherwise $10 = 0
  bnez $10, Lab1 # branch if $10 = 1, i.e., $5 < $6

- There exist pseudo-instructions to help you!
  bit $5, $6, Lab1 # pseudo-instruction translated into
  # at $1.5, 6
  bne $1, $0, 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
  j target # special format for target byte address (26 bits)
  jr $rs # jump to address stored in rt (good for switch statements and transfer tables)

- Call/return functions and procedures
  jal target # jump to target address; save PC of following instruction in rd (aka $ra)
  jr $31 # jump to address stored in $31 (or $ra)

Also possible to use
  jal rd, rd # jumps to address stored in rd; 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
    
    Oper | Offset | Target Offset
    
    beq  | $4.5,1000
    
    bnez | $4,1000

How to address operands

- The ISA specifies addressing modes
- MIPS, as a RISC machine has very few addressing modes
  - Register mode. Operand is in a register
  - Base or displacement or indexed mode
    - Operand is at address 'register + 16-bit signed offset'
  - Immediate mode. Operand is a constant encoded in the instruction
  - PC-relative mode. As base but the register is the PC
Some interesting instructions. Multiply

- Multiplying 2 32-bit numbers yields a 64-bit result
  - Use of Hi and LO registers
    - MulHi rs, rt #Hi/LO = rs*rt
    - MulLo rs, rt
  - Then need to move the Hi or LO or both to regular registers
    - mfi rd #rd = LO
    - mfihi rd #rd = Hi
  - Once more the assembler can come to the rescue with a
    - pseudo inst

Some interesting instructions. Divide

- Similarly, divide needs two registers
  - LO gets the quotient
  - HI gets the remainder
- If an operand is negative, the remainder is not specified by the MIPS ISA.

Logic instructions

- Used to manipulate bits within words, set-up masks etc.
- A sample of instructions
  - and rd, rs, rt #rd = AND(rs, rt)
  - and rd, rs, imm #rd = AND(rs, imm)
  - or rd, rs, rt #rd = OR(rs, rt)
- Immediate constant limited to 16 bits (zero-extended). If longer mask needed, use LUI.
- There is a pseudo-instruction NOT
  - not rd, rs #does 1’s complement (bit by bit
  - complement of rs in rd)

Example of use of logic instructions

- Create a mask of all 1’s for the low-order byte of $6.
  - Don’t care about the other bits.
  - ori $6, $6, 0x00ff
  - ori $6, $6, 0x00ff # $6[7:0] set to 1’s
- Clear high-order byte of register 7 but leave the 3 other bytes unchanged
  - lui $5, 0x0000
  - ori $5, $5, 0xffffffff
  - ori $5, $5, 0xffffffff
  - and $7, $7, $5 # $7 = 0x0000 .... (...whatever was there before)

Shift instructions

- Logical shifts -- Zeros are inserted
  - sll rd, rs, shm #Shift of shm bits; inserting 0’s on the right
  - srl rd, rs, shm #Shift of shm bits; inserting 0’s on the left
- Arithmetic shifts (useful only on the right)
  - sra rd, rs, shm # Sign bit is inserted on the left
- Example let $5 = 0xf000 0000
  - sll $6, $5, 3 # $6 = 0x8000 0000
  - srl $6, $5, 3 # $6 = 0x1fe0 0000
  - sra $6, $5, 3 # $6 = 0xffe0 0000

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]
addd $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)

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00004020</td>
<td>add $8, $0, $0</td>
<td>1: add $8, $0, $0</td>
</tr>
<tr>
<td>0x00004024</td>
<td>ori $9, $0, 5</td>
<td>2: li $9, 5</td>
</tr>
<tr>
<td>0x00004028</td>
<td>ori $1, $0, 4</td>
<td>3: mul $10, $8, $4</td>
</tr>
<tr>
<td>0x0000402c</td>
<td>sw $9, 0($14)</td>
<td>4: sw $9, 0($14)</td>
</tr>
<tr>
<td>0x00004030</td>
<td>sw $9, 0($14)</td>
<td>5: sw $9, 0($14)</td>
</tr>
<tr>
<td>0x00004034</td>
<td>addu $14, $10, $15</td>
<td>6: addu $14, $10, $15</td>
</tr>
<tr>
<td>0x00004038</td>
<td>sw $9, 0($14)</td>
<td>7: sw $9, 0($14)</td>
</tr>
<tr>
<td>0x0000403c</td>
<td>addu $9, $9, $1</td>
<td>8: addu $9, $8, $1</td>
</tr>
<tr>
<td>0x00004040</td>
<td>sw $9, 0($14)</td>
<td>9: sw $9, 0($14)</td>
</tr>
<tr>
<td>0x00004044</td>
<td>sw $9, 0($14)</td>
<td>10: sw $9, 0($14)</td>
</tr>
</tbody>
</table>

10/03/2006  CSE378 lecture recording (c/6)