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,rt,target # go to target if rs = 0
Bne rs,rt,target # go to target if rs ≠ rt
Bltz rs,rt,target # go to target if rs < 0
etc.

Any idea why?

Comparisons between two registers

- Use an instruction to set a third register
  - slt rs,rt,rts # rt = 1 if rs < rt else rt = 0
  - sll rs,rt,k # same but rs and rt are considered unsigned
- Example: Branch to Lab1 if $5 < $6
  - slt $5,$6,$60 # $5 = 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!
  - b $5,$6,Lab1 # pseudo instruction translated into
    - slt $5,$6,$60
    - bnez $10,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 rs (good for switch statements and transfer tables)
- Call/return functions and procedures
  - jal target # jump to target address; save PC of following instruction in $31 (aka $ra)
  - jr $31 # jump to address stored in $31 (or $ra)
  - Also possible to use jal $rd, $ra # jump to address stored in rs, rds=PC of following instruction in rds+0=rd+16-32 bits

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
    Mult r,t,r #HI,LO = rs*rt
  - Multiplies HI or LO or both to regular registers
    mhi rd #rd = LO
    mhlo rd #rd = HI
  - Once the assembler can come to the rescue with a pseudo instruction
    mul rd,rs,rt #generates mult and mhlo
    #and mhlo if necessary

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)
o roi rd,rs,immmed
xor rd,rs,rt
- Immediate constant limited to 16 bits (zero-extended). If longer mask needed, use Lui.
- There is a pseudo-instruction NOT
  not rt,rs #does 1’s complement (bit by bit)
  #complement of rs in rt

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,$0x0000 #8(7,0) set to 1’s
- Clear high-order byte of register 7 but leave the 3 other bytes unchanged
  lui $5,0x0000 #5 = 0x00000000
  ori $5,$5x000000 #5 = 0x00000005
  and $7,$7,$5 #7 = 0x00000007 (... whatever was there before)

Shift instructions

- Logical shifts – Zeros are inserted
  sll rd,rt,sh #left shift of shi bits, inserting 0’s on
  the right
  srl rd,rt,sh #right shift of shi bits; inserting 0’s
  on the left
- Arithmetic shifts (useful only on the right)
  sra rd,rt,sh # Sign bit is inserted on the left
- Example let $5 = 0x0f00 0000
  sll $6,$5,3 #$6 = 0x0f00 0000
  srl $6,$5,3 #$6 = 0x0000 0000
  sra $6,$5,3 #$6 = 0x0000 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  #10 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]
ADDU    $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>0x00400020</td>
<td>ADD $8,0,0</td>
<td>1: add $8,0,0</td>
</tr>
<tr>
<td>0x00400024</td>
<td>ADD $9,0,5</td>
<td>2: li $9,5</td>
</tr>
<tr>
<td>0x00400028</td>
<td>MUL $10,$8,4</td>
<td>3: mul $10,$8,4</td>
</tr>
<tr>
<td>0x00400030</td>
<td>ADDU $14,$10,$15</td>
<td>4: add $14,$10,$15</td>
</tr>
<tr>
<td>0x00400034</td>
<td>SW $9,0($14)</td>
<td>5: sw $9,0($14)</td>
</tr>
<tr>
<td>0x00400038</td>
<td>ADDU $8,$8,1</td>
<td>6: add $8,$8,1</td>
</tr>
<tr>
<td>0x00400040</td>
<td>BLT $8,100,Loop</td>
<td>7: bl $8,100,Loop</td>
</tr>
<tr>
<td>0x00400044</td>
<td>BNE $1,0,-28</td>
<td>8: bne $1,0,-28 [Loop-0x00400044]</td>
</tr>
</tbody>
</table>

6/7/2004    CS378 lab: encoding (v.4)