

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.

Are all of these instructions implemented in the hardware?

Comparisons between two registers

- Use an instruction to set a third register

```
slt      rd,rs,rt      #rd = 1 if rs < rt else rd = 0
sltu     rd,rs,rt      #same but rs 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!

```
blt      $5,$6,Lab1    # pseudo instruction translated into
                        # slt  $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

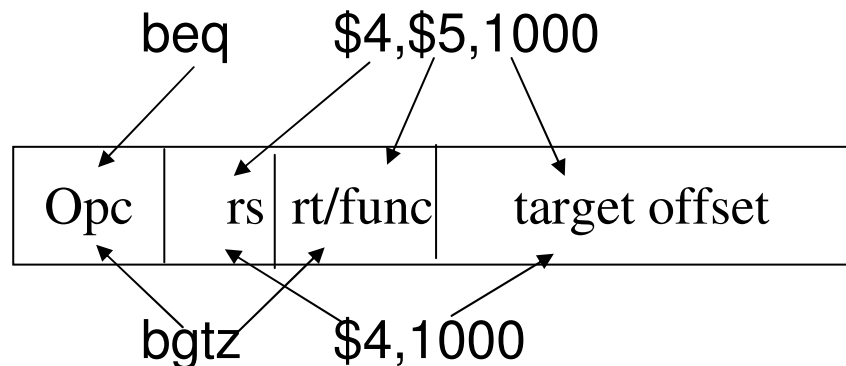
Unconditional transfer of control

- Can use “beqz \$0, target”
 - Very useful but limited range ($\pm 32K$ instructions)
- Use of Jump instructions
 - j target #special format for target inst 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 jalr rs,rd #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 arithmetic instructions



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. 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)

andi rd,rs,immed

or rd,rs,rt

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

- Set the low-order byte of \$6 to all 1's; leave the other bits unchanged

```
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      $5,0x00ff      # $5 = 0x00ff0000
```

```
ori      $5,$5,0xffff   # $5 = 0x00ffffff
```

```
and      $7,$7,$5       # $7 = 0x00..... (...whatever was  
                        #there before)
```

Shift instructions

- Logical shifts -- Zeroes are inserted

sll rd,rt,shm #left shift of shm bits; inserting 0's on
#the right

srl rd,rt,shm #right shift of shm bits; inserting 0's
#on the left

- Arithmetic shifts (useful only on the right)

– sra rd,rt,shm # Sign bit is inserted on the left

- Example let \$5 = 0xff00 0000

sll \$6,\$5,3 # \$6 = 0xf800 0000

srl \$6,\$5,3 # \$6 = 0x1fe0 0000

sra \$6,\$5,3 # \$6 = 0xffe0 0000

Example -- High-level language

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

Assembly language version

Assume: start address of array a in \$15.

We use \$8 to store the value of i and \$9 for the value 5

```
    add    $8,$0,$0    #initialize i
    li     $9,5        # $9 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]
    addiu  $8,$8,1     # increment i
    blt   $8,100,Loop # branch if loop not finished
```

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]
```