

# Logical and Branch Instructions

Tom Kelliher, CS 220

Sept. 5, 2007

## 1 Administrivia

**Announcements**

**Assignment**

Read 2.6.

**From Last Time**

Operands and instruction formats.

**Outline**

1. Logical instructions.
2. Branch and jump instructions.

**Coming Up**

Compiling HLL control structures.

## 2 Last Time

Operations, operands, and instruction formats.

## 3 Logical Operations

The basics:

1. NOT: complement the bits of the operand, bit by bit. ( $\sim$ )
2. AND: AND the bits of two operands, bit by bit. ( $\&$ , not  $\&\&$ ).
3. OR: OR the bits of two operands, bit by bit. ( $|$ , not  $||$ ).
4. Shift: Move the bits of the operand to the left or right a given “distance” ( $\ll$  and  $\gg$ ).  
Complication: logical and arithmetic shifts.

Details:

1. MIPS has no NOT operation, but it does have NOR:  $\sim(a | b)$ .

How do you use NOR to get NOT?

$$\sim 1101 = 0010$$

2.  $1101 \& 1001 = 1001$

and  $\$s2, \$t0 \$t1$

3.  $1001 | 0100 = 1101$

4. Shifts are “similar” to multiplication and division.

$$11001101 \ll 3 = 01101000$$

Usage example: shift and mask operations in finding a character in a word. In C:

```

int charinstr(unsigned char c, unsigned int s)
{
    int i;

    for (i = 0; i < 4; i++)
    {
        if (c == (0xff & s))
            return 1;

        s >>= 8;
    }

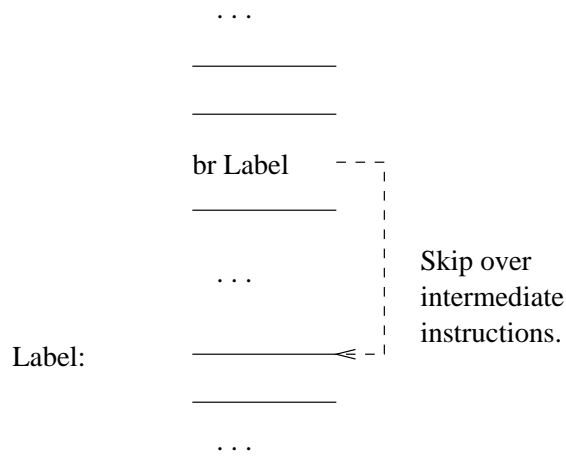
    return 0;
}

charinstr(0xaa, 0xccddaabb);
charinstr(0xaa, 0xbbccdaad);

```

## 4 Branch and Jump Instructions

1. I-format instructions.
2. The idea behind a branch or jump:



3. Branch forward *or* backward  $2^{15}$  **words**.

The complete set, all synthesized from `beq`, `bne`, and `slt`.

Branch instructions use a signed 16-bit offset field; hence they can jump  $2^{15} - 1$  instructions (not bytes) forward or  $2^{15}$  instructions backwards. The *jump* instruction contains a 26 bit address field (the third instruction format).

`b label` *Branch instruction*  
Unconditionally branch to the instruction at the label.

`beq Rsrc1, Src2, label` *Branch on Equal*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` equals `Src2`.

`beqz Rsrc, label` *Branch on Equal Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` equals 0.

`bge Rsrc1, Src2, label` *Branch on Greater Than Equal*  
`bgeu Rsrc1, Src2, label` *Branch on GTE Unsigned*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are greater than or equal to `Src2`.

`bgez Rsrc, label` *Branch on Greater Than Equal Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` are greater than or equal to 0.

`bgt Rsrc1, Src2, label` *Branch on Greater Than*  
`bgtu Rsrc1, Src2, label` *Branch on Greater Than Unsigned*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are greater than `Src2`.

`bgtz Rsrc, label` *Branch on Greater Than Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` are greater than 0.

`ble Rsrc1, Src2, label` *Branch on Less Than Equal*  
`bleu Rsrc1, Src2, label` *Branch on LTE Unsigned*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are less than or equal to `Src2`.

`blez Rsrc, label` *Branch on Less Than Equal Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` are less than or equal to 0.

`blt Rsrc1, Src2, label` *Branch on Less Than*  
`bltu Rsrc1, Src2, label` *Branch on Less Than Unsigned*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are less than `Src2`.

`bltz Rsrc, label` *Branch on Less Than Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` are less than 0.

`bne Rsrc1, Src2, label` *Branch on Not Equal*  
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are not equal to `Src2`.

`bnez Rsrc, label` *Branch on Not Equal Zero*  
Conditionally branch to the instruction at the label if the contents of `Rsrc` are not equal to 0.

`j label` *Jump*  
Unconditionally jump to the instruction at the label.

`jal label` *Jump and Link*  
`jalr Rsrc` *Jump and Link Register*  
Unconditionally jump to the instruction at the label or whose address is in register `Rsrc`. Save the address of the next instruction in register 31.

`jr Rsrc` *Jump Register*  
Unconditionally jump to the instruction whose address is in register `Rsrc`.