Logical and Branch Instructions; Conditional Execution

Tom Kelliher, CS 220

Sept. 23, 2011

1 Administrivia

Announcements

Assignment

From Last Time

Operands and instruction formats.

Outline

1. Logical instructions.
2. Branch and jump instructions.
3. Compiling conditional statements.

Coming Up

Support for procedures.

2 Logical Operations

The basics:

1. NOT: complement the bits of the operand, bit by bit. (`~`)
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" (`<<` and `>>`). Complication: logical and arithmetic shifts.

Details:

1. MIPS has no NOT operation, but it does have NOR: `~(a | b)`.

   How do you use NOR to get NOT?

   `~1101 = 0010`

2. `1101 & 1001 = 1001`

   `and $s2, $t0, $t1`

3. `1001 | 0100 = 1101`

4. Shifts are "similar" to multiplication and division.

   `11001101 << 3 = 01101000`

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

```
int charinword(unsigned char c, unsigned int w)
{
   int i;

   for (i = 0; i < 4; i++)
   {
```

```
      if (c == (0xff & w))
         break;

      w = w >> 8;
   }

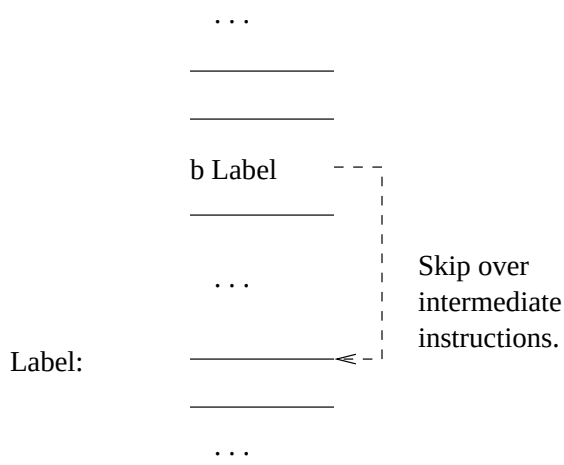
   return 0;
}

charinword(0xaa, 0xccddaabb);
charinword(0xaa, 0xbbccdaad);
```

3 Branch and Jump Instructions

1. I-format instructions.

2. The idea behind a branch or jump:



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

All branch instructions are 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).

Summary of branch instructions:

1. Unconditional branch: `b label`

   Example:

```
          b foo
          ...
foo:      add $1, $1, $1
```

2. One operand branches:

   (a) The list: `beqz`, `bnez`, `bgez`, `bgtz`, `blez`, `bltz`.

   (b) Example: `bnez $s0, label`

3. Two operand branches:

   (a) The list: `beq`, `bne`, `bge`, `bgt`, `ble`, `blt`.

   Unsigned versions.

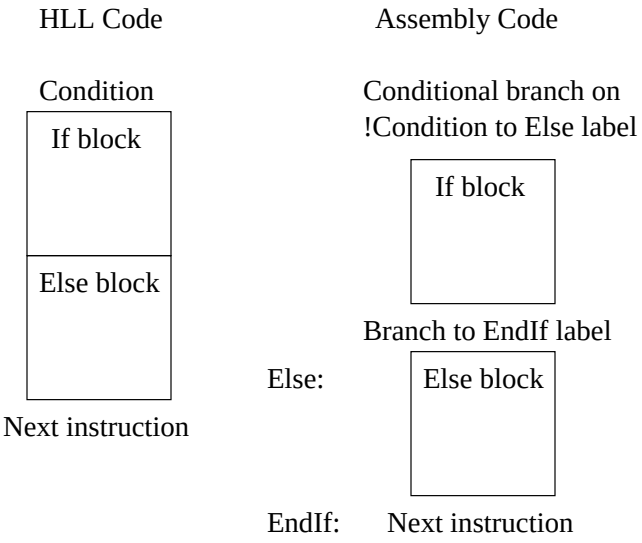
   (b) Second comparison operand may be a register or a constant:

```
bge $sp, $ra, foo
blt $s0, 5, bar
```

4 Compiling HLL Control Structures

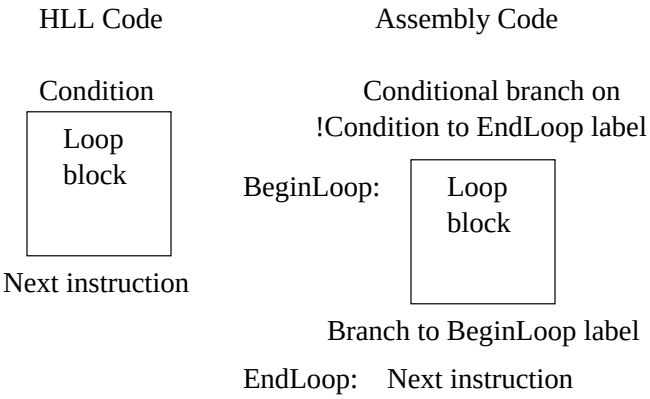
Write MIPS code fragments corresponding to the following:

1. Compiling an if:




```
if (i < 12)
   ++i;
else
   --j;
```

2. Compiling a loop:




```
i = 1;
j = 0;
while (i < 200)
{
   j += i;
   i *= i;
}
```

5 Class Assignment

Write MIPS code corresponding/solving each of the following:

1. ```
   j = 0;
   for (i = 0; i < 10; ++i)
      j += i;
   ```

2. ```
   j = 0;
   for (i = 0; i < 10; ++i)
      if (i > 5)
         j += i;
   ```

3. ```
   while (i > 0 && i < 10)
      ++i;
   ```

4. ```
   if (i < 12 && j > 3 || k != 0)
      ++i;
   else if (i == 33)
      --j;
   else
      k += 2;
   ```

5. ```
   int i;

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

      w = w >> 8;
   }
   ```

6. (3.9 from the 3rd edition of the text) The naive way of compiling

   ```
   while (save[i] == k)
      i += k;
   ```

   requires execution of both a conditional branch and an unconditional jump each time through the loop. Produce the naive code.

   Optimize the naive code so that only a conditional branch is executed each time through the loop.
   ```

7. (3.24 from the 3rd edition of the text, a variation) Write a code segment which takes two "parameters:"

   (a) An ASCII character in `$a0`.

   (b) A pointer to a NULL-terminated string in `$a1`.

   and "returns" a count of the number of occurrences of the character in the string in `$v0`.