Conditional Execution I

Tom Kelliher, CS 240

Feb. 4, 2002

1 Administrivia

Announcements

Assignment

Nothing new.

From Last Time

Operands and instruction formats.

Outline

1. Immediate operands.
2. Branch and jump instructions.
3. Compiling HLL control structures.
4. Class teamwork assignment.

Coming Up

Conditional Execution II.

2 Immediate Operands

1. Operand types (addressing modes) we've seen so far: registers, memory.

2. What about constants? Where have we already seen immediates? Arithmetic example:

```
addi $t0, $s0, 8   # An immediate operand.
```

   Why no `subi`?

3. Immediate operand: found within the instruction itself.

4. Small immediates occur frequently, so...

5. Design principle 4: Make the common case fast.

6. But, how do I load a 32-bit immediate? `lui` followed by `addi` (whoops, sign extension) or `ori`:

```
lui $s0, 0x5555
ori $s0, $s0, 0xaaaa
```

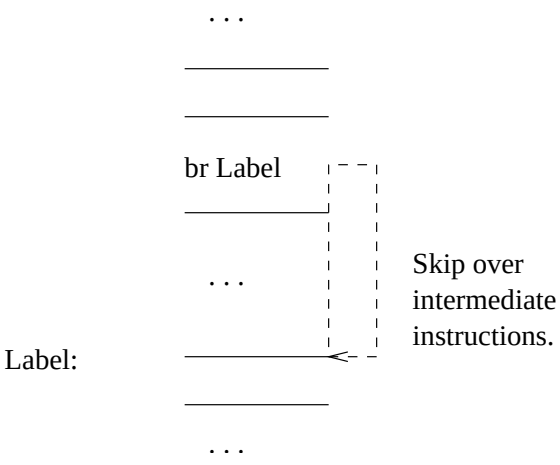
7. How does the assembler manufacture 32-bit immediates for us? Register `$at`.

8. Programmer conventions. "Enhanced" assembly language. Simplicity.

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**.

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.

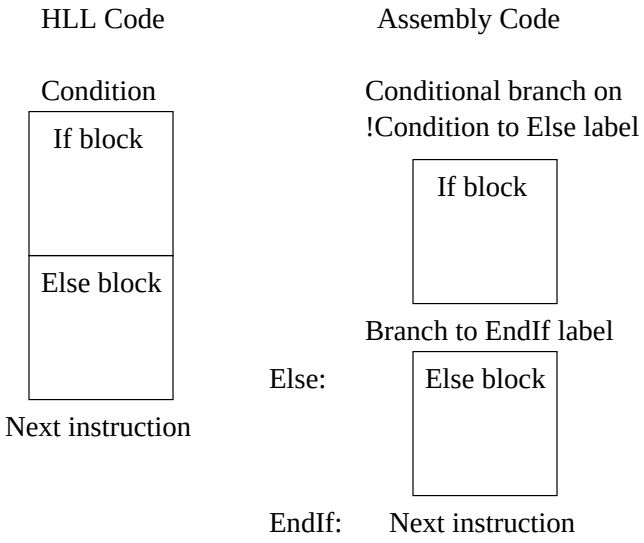
`jr Rsrc` *Jump Register*

Unconditionally jump to the instruction whose address is in register `Rsrc`.

4 Compiling HLL Control Structures

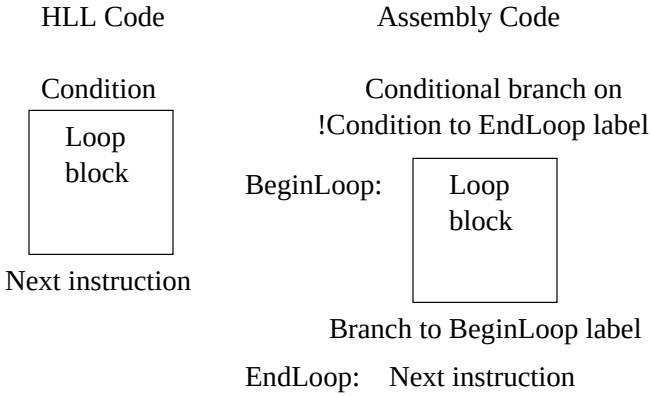
Write MIPS code fragments corresponding to the following:

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 Teamwork Assignment

The class, working as a team, is to e-mail the solution to the following problems to `GC_CS_240_001_SPRING_2002@gnav.goucher.edu`. Let me know who participated in the solution of what problem(s).

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. (3.9 from 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.

6. (3.24 from 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`.