Conditional Execution

Tom Kelliher, CS 220
Dec. 8, 2003

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.
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:

   \[
   \text{addi } $t0, $s0, 8 \quad \# \text{ An immediate operand.}
   \]

   Why no \text{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? \text{lui} followed by \text{addi} (whoops, sign extension) or \text{ori}:

   \[
   \begin{align*}
   \text{lui } & $s0, 0x5555 \\
   \text{ori } & $s0, $s0, 0xaaaa
   \end{align*}
   \]

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


3 Branch and Jump Instructions

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

```
... 
------
------
br Label |
------
... 
... 
Skip over intermediate instructions.
Label:  
------
... 
```

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 Rsrb1, Src2, label 
```

*Branch on Equal*

Conditionally branch to the instruction at the label if the contents of register `Rsrb1` equals `Src2`.

```
beqz Rsrb, label 
```

*Branch on Equal Zero*

Conditionally branch to the instruction at the label if the contents of `Rsrb` equals 0.

```
bge Rsrb1, Src2, label 
bgeu Rsrb1, Src2, label 
```

*Branch on Greater Than Equal*

*Branch on GTE Unsigned*

Conditionally branch to the instruction at the label if the contents of register `Rsrb1` are greater than or equal to `Src2`.
bgez Rs, label
Conditionally branch to the instruction at the label if the contents of Rs are greater than or equal to 0.

bgt Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are greater than Rs2.

bgtu Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are greater than or equal to Rs2.

bgz Rs, label
Conditionally branch to the instruction at the label if the contents of Rs are greater than 0.

ble Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are less than or equal to Rs2.

bleu Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are less than or equal to Rs2.

blez Rs, label
Conditionally branch to the instruction at the label if the contents of Rs are less than or equal to 0.

blt Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are less than Rs2.

bltu Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are less than or equal to Rs2.

bltz Rs, label
Conditionally branch to the instruction at the label if the contents of Rs are less than 0.

bne Rs1, Rs2, label
Conditionally branch to the instruction at the label if the contents of register Rs1 are not equal to Rs2.

bnez Rs, label
Conditionally branch to the instruction at the label if the contents of Rs are not equal to 0.
### 4 Compiling HLL Control Structures

Write MIPS code fragments corresponding to the following:

1. Compiling an if:

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

   - **Assembly Code**
     
     ```
     Condition
     If block
     Else block
     Next instruction
     Conditional branch on
     !Condition to Else label
     Branch to EndIf label
     Else:
     Else block
     EndIf: Next instruction
     ```
2. Compiling a loop:

<table>
<thead>
<tr>
<th>HLL Code</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Conditional branch on</td>
</tr>
<tr>
<td>Loop block</td>
<td>!Condition to EndLoop label</td>
</tr>
<tr>
<td>Next instruction</td>
<td>BeginLoop: Loop block</td>
</tr>
<tr>
<td></td>
<td>Branch to BeginLoop label</td>
</tr>
<tr>
<td></td>
<td>EndLoop: Next instruction</td>
</tr>
</tbody>
</table>

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

5 Class Teamwork Assignment

Working in groups of 2–3, solve as many of the following as possible. Turn in your solutions.

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 $v0$. 