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:

   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.


3 Branch and Jump Instructions

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

```
... _____
_____  
br Label    _____
...     Skip over
...     intermediate
... 
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`  
  Unconditionally branch to the instruction at the label.

- `beq Rsnc1, Src2, label`  
  Conditionally branch to the instruction at the label if the contents of register `Rsnc1` equals `Src2`.

- `beqz Rsnc, label`  
  Conditionally branch to the instruction at the label if the contents of `Rsnc` equals 0.

- `bge Rsnc1, Src2, label`  
  Conditionally branch to the instruction at the label if the contents of register `Rsnc1` are greater than or equal to `Src2`.

- `bgeu Rsnc1, Src2, label`  
  Conditionally branch to the instruction at the label if the contents of register `Rsnc1` are greater than or equal to `Src2`.  

Branch instruction

Branch on Equal

Branch on Equal Zero

Branch on Greater Than Equal

Branch on GTE Unsigned
\textbf{bgez} Rsrc, label \quad \textit{Branch on Greater Than Equal Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are greater than or equal to 0.

\textbf{bgt} Rsrc1, Src2, label \quad \textit{Branch on Greater Than}
\textbf{bgtu} Rsrc1, Src2, label \quad \textit{Branch on Greater Than Unsigned}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are greater than \texttt{Src2}.

\textbf{bgtz} Rsrc, label \quad \textit{Branch on Greater Than Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are greater than 0.

\textbf{ble} Rsrc1, Src2, label \quad \textit{Branch on Less Than Equal}
\textbf{bleu} Rsrc1, Src2, label \quad \textit{Branch on LTE Unsigned}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are less than or equal to \texttt{Src2}.

\textbf{blez} Rsrc, label \quad \textit{Branch on Less Than Equal Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are less than or equal to 0.

\textbf{blt} Rsrc1, Src2, label \quad \textit{Branch on Less Than}
\textbf{bltu} Rsrc1, Src2, label \quad \textit{Branch on Less Than Unsigned}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are less than \texttt{Src2}.

\textbf{bltz} Rsrc, label \quad \textit{Branch on Less Than Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are less than 0.

\textbf{bne} Rsrc1, Src2, label \quad \textit{Branch on Not Equal}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are not equal to \texttt{Src2}.

\textbf{bnez} Rsrc, label \quad \textit{Branch on Not Equal Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are not equal to 0.
j label
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:

<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>If block</td>
<td>!Condition to Else label</td>
</tr>
<tr>
<td>Else block</td>
<td>Branch to EndIf label</td>
</tr>
<tr>
<td>Next instruction</td>
<td>Else: Else block</td>
</tr>
<tr>
<td></td>
<td>EndIf: Next instruction</td>
</tr>
</tbody>
</table>

   ```
   if (i < 12)
     ++i;
   else
     --j;
   ```
2. Compiling a loop:

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

```
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 $v0.$