Agenda

- Review of last lecture (Control Flow Instructions)
- More of Control Flow Instructions
- Instruction encoding
- Addressing modes
Conditional Operations

• Branch to a labeled instruction if a condition is true
  – Otherwise, continue sequentially

• beq rs, rt, L1
  – if (rs == rt) branch to instruction labeled L1;

• bne rs, rt, L1
  – if (rs != rt) branch to instruction labeled L1;

• j L1
  – unconditional jump to instruction labeled L1
Signed vs. Unsigned

• Signed comparison: `slt`, `slti`
• Unsigned comparison: `sltu`, `sltui`
• Example
  – $s0 = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$
  – $s1 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001$
  – `slt $t0, s0, s1`  # signed
    • $-1 < +1 \Rightarrow t0 = 1$
  – `sltu $t0, s0, s1`  # unsigned
    • $+4,294,967,295 > +1 \Rightarrow t0 = 0$
MIPS PC-relative or branch addressing

• Branch instructions specify
  – Opcode, two registers, target address
• Most branch targets are near branch
  – Forward or backward

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>constant or address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

- **PC-relative addressing**
  - Target address = PC + offset × 4
  - PC already incremented by 4 by this time
Pseudodirect or Jump Addressing

- Jump (j and jal) targets could be anywhere in text segment
  - Encode full address in instruction

<table>
<thead>
<tr>
<th>op</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

- (Pseudo)Direct jump addressing
  - Target address = PC\textsubscript{31...28} : (address × 4)
Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>0</th>
<th>0</th>
<th>19</th>
<th>9</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>sll $t1, $s3, 2</td>
<td>80000</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>add $t1, $t1, $s6</td>
<td>80004</td>
<td>0</td>
<td>9</td>
<td>22</td>
<td>9</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>lw $t0, 0($t1)</td>
<td>80008</td>
<td>35</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne $t0, $s5, Exit</td>
<td>80012</td>
<td>5</td>
<td>8</td>
<td>21</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi $s3, $s3, 1</td>
<td>80016</td>
<td>8</td>
<td>19</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>j Loop</td>
<td>80020</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exit: ... 80024
Note on the PC incrementing

• Technical term for auto-incrementation of PC is “delayed branch”

• By default in SPIM “delayed branch” is not checked. To see your SPIM settings look at simulator → settings

• You can also check it by loading code to SPIM to check

```
main : bne $s0, $s0, main
```
Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

```assembly
beq $s0, $s1, L1
↓
bne $s0, $s1, L2
j L1
L2: ...
```
CPU overview
## Addressing Modes

<table>
<thead>
<tr>
<th>Addressing</th>
<th>Instruction</th>
<th>Other elements involved</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td></td>
<td>Some place in the machine</td>
<td></td>
</tr>
<tr>
<td>Immediate</td>
<td></td>
<td>Extend, if required</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Reg spec</td>
<td>Reg file</td>
<td>Reg data</td>
</tr>
<tr>
<td>Base</td>
<td>Constant offset</td>
<td>Add</td>
<td>Memory</td>
</tr>
<tr>
<td>PC-relative</td>
<td>Constant offset</td>
<td>Add</td>
<td>Memory</td>
</tr>
<tr>
<td>Pseudodirect</td>
<td>PC</td>
<td>Mem addr</td>
<td>Memory</td>
</tr>
</tbody>
</table>

Schematic representation of addressing modes in MIPS.
More Elaborate Addressing Modes

Addressing | Instruction | Other elements involved | Operand
---|---|---|---
Indexed | | |
Update (with base) | | |
Update (with indexed) | | |
Indirect | | |

Schematic representation of more elaborate addressing modes not supported in MIPS.
Example: Basic x86 Addressing Modes

- Two operands per instruction

<table>
<thead>
<tr>
<th>Source/dest operand</th>
<th>Second source operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Immediate</td>
</tr>
<tr>
<td>Register</td>
<td>Memory</td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
</tr>
<tr>
<td>Memory</td>
<td>Immediate</td>
</tr>
</tbody>
</table>

- Memory addressing modes
  - Address in register
  - Address = \( R_{\text{base}} + \text{displacement} \)
  - Address = \( R_{\text{base}} + 2^{\text{scale}} \times R_{\text{index}} \) (scale = 0, 1, 2, or 3)
  - Address = \( R_{\text{base}} + 2^{\text{scale}} \times R_{\text{index}} + \text{displacement} \)
Usefulness of Some Elaborate Addressing Modes

Update mode: XORing a string of bytes

```
loop:  lb   $t0,A($s0)
xor  $s1,$s1,$t0
addi $s0,$s0,-1
bne  $s0,$zero,loop
```

One instruction with update addressing

Indirect mode: Case statement

```
case: lw  $t0,0($s0)  # get s
      add  $t0,$t0,$t0  # form 2s
      add  $t0,$t0,$t0  # form 4s
      la   $t1,T        # base T
      add  $t1,$t0,$t1
      lw   $t2,0($t1)   # entry
      jr   $t2
```

Branch to location $L_i$ if $s = i$ (switch var.)

<table>
<thead>
<tr>
<th>Branch Address</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>L0</td>
</tr>
<tr>
<td>T+4</td>
<td>L1</td>
</tr>
<tr>
<td>T+8</td>
<td>L2</td>
</tr>
<tr>
<td>T+12</td>
<td>L3</td>
</tr>
<tr>
<td>T+16</td>
<td>L4</td>
</tr>
<tr>
<td>T+20</td>
<td>L5</td>
</tr>
</tbody>
</table>
Example of decoding and addressing modes in datapath
Let’s add more details on this figure to see why instruction decoding could be simple and to see what is happening with for different instructions.
R-Type Instruction
Load Instruction
Branch-on-Equal Instruction
Implementing Jumps

<table>
<thead>
<tr>
<th>Jump</th>
<th>2</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31:26</td>
<td>25:0</td>
</tr>
</tbody>
</table>

- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode
Datapath With Jumps Added
**Review**

- **Instructions so far:**
  
  - add, addi, sub, xor, or, and, sll, srl, sla, sllv, srlv
  - lw, sw, lb, lbu, lh, lhu
  - beq, bne, j, jal, jr
  - slt, slti, sltu, sltui

- **Registers so far**
  
  - C variables: $s0 - s7$
  - Temporary variables: $t0 - t9$
  - Zero: $zero$