Computer Organization: A real processor

Steven Balensiefer

Background
- We built a model processor
  - You get to make it work
- Heavily based on MIPS2000
  - Described by Patterson & Hennessy
- Single-cycle design
  - All operations take 1 (long) cycle

Instruction Set Specs
- 32 registers
- Load-Store Architecture
- Word Addressing
- 3 Formats for Instructions
  - Register to Register
  - Immediate
  - Jump

Instruction Encodings
- R-format: $r3 = r1 + r2$
  | op | rs | rt | rd | shft | func |
  | ALU | r1 | r2 | r3 | X | ADD |
- I-format: $r3 = imm(r2)$
  | op | rs | rt | addr/immediate |
  | Load | r2 | r3 | imm |
- J-format: $jal hanoi$
  | op | target address |
  | JAL | addr(hanoi) |
Controller

Skeleton Code:

ADDI: begin
    wrDataSel = 2'b0x;
    mW = 1'b0;
    mR = 1'b0;
    PCSel = 2'b0x;
    srcB = 1'b0;
    regWrite = 1'b0;
    wrRegSel = 2'b0x;
    op = 6'b000000;
end
### ALU

\[
\text{assign B} = \text{select between \( R_{\text{reg}} \) and \( \text{imm} \)}; \\
\text{always} \{0(A \text{ or } B \text{ or } \text{op}) \text{ begin} \\
\text{case (op)} \\
0 \{00000001: \text{result} = A + B; \} \\
\text{default: \text{result} = 32'h00000000} \}; \\
\text{end}
\]

### Data Memory

- **Address from ALU**
- **Data from Reg B**
- **Memory-Mapped I/O**
  - SW to xFFFFFFF
  - Buffers / Displays

(See dmemory.v for more)

### Miscellaneous

**WrRegSel:**
- 00: Out = ALU
- 01: Out = MEM
- 1X: Out = PC + 1

**SignExtender:**
- convert 16-bit to 32-bit
### R-Format Operations

<table>
<thead>
<tr>
<th>Func</th>
<th>Operation</th>
<th>PC</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>rd = rs + rt</td>
<td>PC++</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>rd = rs – rt</td>
<td>PC++</td>
<td></td>
</tr>
<tr>
<td>SLT</td>
<td>rd = (rs &lt; rt) ? 1 : 0</td>
<td>PC++</td>
<td>Set on less than</td>
</tr>
<tr>
<td>JR</td>
<td>No change</td>
<td>PC=rs</td>
<td>Jump to Register</td>
</tr>
</tbody>
</table>

### I-Format Ops:

<table>
<thead>
<tr>
<th>Operation</th>
<th>PC</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>rt = rs+SE(imm)</td>
<td>PC++</td>
</tr>
<tr>
<td>ORI</td>
<td>rt = rs</td>
<td>imm</td>
</tr>
<tr>
<td>LUI</td>
<td>rt = imm &lt;&lt; 16</td>
<td>PC++</td>
</tr>
<tr>
<td>Load upper immed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>rt = MEM[rs+se(imm)]</td>
<td>PC++</td>
</tr>
<tr>
<td>SW</td>
<td>MEM[rs+se(imm)] = rt</td>
<td>PC++</td>
</tr>
<tr>
<td>BEQ</td>
<td>(rs == rt)?</td>
<td>PC+1+(0</td>
</tr>
</tbody>
</table>

### J-Format Ops:

<table>
<thead>
<tr>
<th>Func</th>
<th>Operation</th>
<th>PC</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td></td>
<td>PC = target</td>
<td>a.k.a GOTO</td>
</tr>
<tr>
<td>JAL</td>
<td>r31 = PC+1</td>
<td>PC = target</td>
<td>Jump and Link</td>
</tr>
</tbody>
</table>

JAL stores next address, jumps to target (a.k.a fn call)

### Final Tips:

- Verilog uses “?” for Don’t Cares
- Waveforms will make things easier
- Be sure to set clk and reset
**Programming Example**

Given: A is an array of size B

Goal: Compute $\sum_{i=0}^{B} A[i]$

I’ll just use a for loop …

---

**Assembly Language**

- Variables → Registers
- Array Access → Load (name+offest)
- Minimal Control Structures
  - Branches (A < B, A >= B, A != B)
  - Jumps

---

**C to ASM**

<table>
<thead>
<tr>
<th>High Level Language</th>
<th>Pseudo-Asm</th>
</tr>
</thead>
<tbody>
<tr>
<td>C = 0; for(i = 0; i &lt; B; i = i+1) { C = C + A[i]; }</td>
<td>C = 0; Loop: bge i, B, Exit temp = A+i temp2 = load 0(temp) C = C + temp2; i = i + 1; j Loop Exit: ...</td>
</tr>
</tbody>
</table>

---

**ASM to RTL**

C = 0; i = 0; Loop: bge i, B, Exit temp = load A[i]; C = C + temp; i = i + 1; j Loop Exit: ...

r3 = r0, PC++; r4 = r0, PC++; Loop: PC = (r4 ≥ r2) ? Exit : PC+1

r6 = MEM[r4+r1 ],PC++; r3 = r3 + r6, PC++; r4 = r4 + 1, PC++; PC = LOOP Exit: