Anatomy: 5 components of any Computer

- Personal Computer
- Processor
- Control ("brain")
- Datapath ("brawn")
- Memory

This week

Review: A Single Cycle Datapath

- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals

An Abstract View of the Critical Path

- Critical Path (Load Operation) = Delay clock through PC (FFs) + Instruction Memory’s Access Time + Register File’s Access Time, + ALU to Perform a 32-bit Add + Data Memory Access Time + Stable Time for Register File Write

Recap: Meaning of the Control Signals

- \(n\) = next
- Later in lecture: higher-level connection between mux and branch cond

Recap: Meaning of the Control Signals

- ExtOp: "zero", "sign"  
- ALUsrc: 0 => regB; 1 => immmed  
- ALUctr: "add", "sub", "or"  
- MemWr: 1 => write memory  
- RegDst: 0 => "rt"; 1 => "rd"  
- MemtoReg: 0 => ALU; 1 => Memory  
- RegWr: 1 => write register
### RTL: The Add Instruction

<table>
<thead>
<tr>
<th>Opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>061116212631</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**add rd, rs, rt**

- **MEM(PC)**: Fetch the instruction from memory
- **R[rd] = R[rs] + R[rt]**: The actual operation
- **PC = PC + 4**: Calculate the next instruction’s address

### Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory: Instruction = MEM[PC]

### The Single Cycle Datapath during Add

- **R[rd] = R[rs] + R[rt]**

### Instruction Fetch Unit at the End of Add

- **PC = PC + 4**

### Single Cycle Datapath during Or Immediate?

- **R[rt] = R[rs] OR ZeroExt[Imm16]**
The Single Cycle Datapath during Load?

- \( R[rt] = \text{Data Memory} \left( R[rs] + \text{SignExt}[imm16] \right) \)

The Single Cycle Datapath during Store?

- Data Memory \( \{ R[rs] + \text{SignExt}[imm16] \} = R[rt] \)

The Single Cycle Datapath during Branch?

- if \( (R[rs] - R[rt] == 0) \) then \( \text{Zero} = 1 \); else \( \text{Zero} = 0 \)
Instruction Fetch Unit at the End of Branch

- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4
- else PC = PC + 4

- What is encoding of nPC_sel?
- Direct MUX select?
- Branch / not branch
- Let’s pick 2nd option

A Summary of the Control Signals (1/2)

<table>
<thead>
<tr>
<th>Inst</th>
<th>Register Transfer</th>
<th>ALUctrRegDst</th>
<th>ALUsrcExtOp</th>
<th>MemtoRegMemWr</th>
<th>Zero</th>
<th>nPC_sel</th>
<th>addr</th>
<th>sub</th>
<th>srl</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[r] &lt;- R[s] + R[t]</td>
<td>PC &lt;- PC + 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUctr</td>
<td>RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUSrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>R[r] &lt;- R[r] - R[t]</td>
<td>PC &lt;- PC + 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ</td>
<td>if (R[r] == R[t]) then PC &lt;- PC + sign_ext(imm16) ]</td>
<td></td>
<td>00 else PC &lt;- PC + 4</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A Summary of the Control Signals (2/2)

<table>
<thead>
<tr>
<th>Type</th>
<th>op</th>
<th>target address</th>
<th>J-type jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target address</td>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 4: Given Datapath: RTL -> Control

<table>
<thead>
<tr>
<th>Instruction Fetch Unit</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Inst</th>
<th>RegWr</th>
<th>RegDst</th>
<th>ALUctr</th>
<th>MemtoRegMemWr</th>
<th>Zero</th>
<th>nPC_sel</th>
<th>addr</th>
<th>sub</th>
<th>srl</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[r]</td>
<td>R[s]</td>
<td>R[t]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>R[r]</td>
<td>R[r]</td>
<td>R[t]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ</td>
<td>if (R[r] == R[t]) then PC &lt;- PC + sign_ext(imm16) ]</td>
<td></td>
<td>00 else PC &lt;- PC + 4</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td>ALUctr = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Single Cycle Datapath during Jump

- New PC = { PC[31..28], target address, 00 }
The Single Cycle Datapath during Jump

- New PC = \{ PC[31..28], target address, 00 \}

Instruction Fetch Unit at the End of Jump

- New PC = \{ PC[31..28], target address, 00 \}

Instruction Fetch Unit at the End of Jump

- New PC = \{ PC[31..28], target address, 00 \}

Build CL to implement Jump on paper now

- New PC = \{ PC[31..28], target address, 00 \}

Build CL to implement Jump on paper now

- New PC = \{ PC[31..28], target address, 00 \}

Peer Instruction

- New PC = \{ PC[31..28], target address, 00 \}

A. MemToReg='x' & ALUctr='sub'. SUB or BEQ?

B. ALUctr='add'. Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?

C. “Don’t Care” signals are useful because we can simplify our Boolean equations?
And in Conclusion… Single cycle control

5 steps to design a processor

1. Analyze instruction set => datapath requirements
2. Select set of datapath components & establish clock methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic

Control is the hard part

MIPS makes that easier

- Instructions same size
- Source registers always in same place
- Immediates same size, location
- Operations always on registers/immediates