**Review: Single cycle datapath**

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 transfser.
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

---

### Single Cycle Datapath during Or Immediate?

- \( R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{imm16}] \)

### Single Cycle Datapath during Load?

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

---

Soon after delivering a 10,240 processor supercomputer to NASA, SGI delivers a 2,048 node system to Japan with the worlds largest memory capacity, 13TB

MIPS makes that easier:

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

---

**Operations always on registers/immediates**

---

**The Single Cycle Datapath during Load?**

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

---

**The Single Cycle Datapath during Or Immediate?**

- \( R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{imm16}] \)
The Single Cycle Datapath during Store

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

\[
\begin{array}{c}
\text{aluctr} = \text{Clk} \\
\text{busW} = R[rt] \\
\text{RegWr} = 0 \\
\text{busA} = R[rs] \\
\text{busB} = \text{imm16} \\
\text{Rw} = \text{Ra} = \text{Rb} = 32 \\
\end{array}
\]

\[
\begin{array}{c}
\text{RegDst} = x \\
\text{ExtOp} = 1 \\
\text{MemtoReg} = x \\
\text{Zero} = 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{ExtOp} = 1 \\
\text{Data In} = \text{Instr} < 31:0 > \\
\text{WrEn} = 0 \\
\text{Adr} = \text{imm16} \\
\text{Data} = \\
\text{Memory} = 32 \\
\text{MemWr} = 0 \\
\text{MemReg} = x \\
\end{array}
\]

\[
\begin{array}{c}
\text{alu} = \text{Add} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Add} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Add} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Add} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\end{array}
\]

The Single Cycle Datapath during Branch

- if \( (R[rs] - R[rt] = 0) \) then Zero = 1; else Zero = 0

\[
\begin{array}{c}
\text{aluctr} = \text{Sub} \\
\text{busW} = R[rt] \\
\text{RegWr} = 0 \\
\text{busA} = R[rs] \\
\text{busB} = \text{imm16} \\
\text{Rw} = \text{Ra} = \text{Rb} = 32 \\
\end{array}
\]

\[
\begin{array}{c}
\text{RegDst} = x \\
\text{ExtOp} = 0 \\
\text{MemtoReg} = x \\
\text{Zero} = 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{ExtOp} = 0 \\
\text{Data In} = \text{Instr} < 31:0 > \\
\text{WrEn} = 0 \\
\text{Adr} = \text{imm16} \\
\text{Data} = \\
\text{Memory} = 32 \\
\text{MemWr} = 0 \\
\text{MemReg} = x \\
\end{array}
\]

\[
\begin{array}{c}
\text{alu} = \text{Sub} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Sub} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Sub} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\text{alu} = \text{Sub} \\
\text{aluSec} = \\
\text{busA} = \text{Data} \\
\text{busB} = \text{Data} \\
\text{RegWr} = 0 \\
\text{aluctr} = \text{Clk} \\
\text{aluSec} = \\
\end{array}
\]

Instruction Fetch Unit at the End of Branch

- if \( (\text{Zero} == 1) \) then \( \text{PC} = \text{PC} + 4 + \text{SignExt}[\text{imm16}]*4 \) ; else \( \text{PC} = \text{PC} + 4 \)

\[
\begin{array}{c}
\text{nPCMUXsel} = \text{Br} \\
\text{nPCMUXsel} = \text{Br} \\
\end{array}
\]

Step 4: Given Datapath: RTL -> Control

- What is encoding of \( \text{nPCMUXsel} \)?
- Direct MUX select?
- Branch / not branch
- Let’s pick 2nd option

\[
\begin{array}{c}
\text{Q: What logic gate?} \\
\text{MUX} \\
\end{array}
\]
A Summary of the Control Signals (1/2)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register Transfer</th>
<th>ALU src</th>
<th>ALU control</th>
<th>Mem to Reg</th>
<th>Register</th>
<th>ALU write</th>
<th>nPC select</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[rd] ← R[rs] + R[rt]; PC ← PC + 4</td>
<td>RegB</td>
<td>&quot;add&quot;</td>
<td>+4</td>
<td>rd</td>
<td>rd</td>
<td>RDST</td>
</tr>
<tr>
<td>SUB</td>
<td>R[rd] ← R[rs] - R[rt]; PC ← PC + 4</td>
<td>RegB</td>
<td>&quot;sub&quot;</td>
<td>+4</td>
<td>rd</td>
<td>rd</td>
<td>RDST</td>
</tr>
<tr>
<td>ORI</td>
<td>R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4</td>
<td>Im</td>
<td>&quot;or&quot;</td>
<td>+4</td>
<td>rt</td>
<td>rt</td>
<td>RDST</td>
</tr>
<tr>
<td>LOAD</td>
<td>R[rs] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4</td>
<td>Im</td>
<td>&quot;add&quot;</td>
<td>+4</td>
<td>rs</td>
<td>rs</td>
<td>RDST</td>
</tr>
<tr>
<td>STORE</td>
<td>MEM[ R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4</td>
<td>Im</td>
<td>&quot;add&quot;</td>
<td>+4</td>
<td>rs</td>
<td>rs</td>
<td>RDST</td>
</tr>
</tbody>
</table>

A Summary of the Control Signals (2/2)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>ALU src</th>
<th>ALU control</th>
<th>Mem to Reg</th>
<th>Register</th>
<th>ALU write</th>
<th>nPC select</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ</td>
<td>if ( R[rs] == R[rt] ) then PC ← PC + sign_ext(Imm16)</td>
<td></td>
<td>00 else PC ← PC + 4</td>
<td></td>
<td>&quot;sub&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Adminivia

- Final exam time/location set
  - Tuesday, December 14th, 12:30 – 3:30 pm
  - At the Hearst Gym (lucky us!)

Finite State Machines extremely useful!

- They define
  - How output signals respond to input signals and previous state.
  - How we change states depending on input signals and previous state.
- The output signals could be our familiar control signals
  - Some control signals may only depend on CL, not on state at all...
- We could implement very detailed FSMs with Programmable Logic Arrays

Review: Finite State Machine (FSM)

- States represent possible output values.
- Transitions represent changes between states based on inputs.
- Implement with CL and clocked register feedback.

Taking advantage of sum-of-products

- Since sum-of-products is a convenient notation and way to think about design, offer hardware building blocks that match that notation
- One example is Programmable Logic Arrays (PLAs)
- Designed so that can select (program) ands, ors, complements after you get the chip
  - Late in design process, fix errors, figure out what to do later, ...
Programmable Logic Arrays

- Pre-fabricated building block of many AND/OR gates
- "Programmed" or "Personalized" by making or breaking connections among gates
- Programmable array block diagram for sum of products form

And Programming:
- How many inputs?
- How many outputs?
- How to combine inputs?
- How to combine product terms?

Enabling Concept

- Shared product terms among outputs

Example:

F₀ = A     +  B' C'
F₁ = A C'  +  A B
F₂ = B' C' +  A B
F₃ = B' C  +  A

Personality matrix

<table>
<thead>
<tr>
<th>Product terms</th>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B' C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A C'</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B'C'</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Before Programming

- All possible connections available before "programming"

After Programming

- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)

Alternate Representation

- Short-hand notation--don't have to draw all the wires
  - X Signifies a connection is present and perpendicular signal is an input to gate

Other Programmable Logic Arrays

- There are other types of PLAs which can be reprogrammed on the fly
- The most common is called a Field Programmable Gate Array (FPGA)
- FPGAs are made up of configurable logic blocks (CLBs) and flip-flops which can be programmed by software
- Berkeley has on-going research into reconfigurable computing with FPGAs
  - Check out Brass and BEE2 projects
Peer Instruction

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 PLA personality matrix. F / T?

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
• Immediate same size, location
• Operations always on registers/immediates

And in Conclusion... Single cycle control