**CS 61C: Great Ideas in Computer Architecture (Machine Structures)**

**MIPS Control, MIPS Pipelining Intro**

Instructor: Michael Greenbaum

---

**Review: A Single Cycle Datapath**

- Instruction: \(<31:0>\>

- **Architectural Implementation**
  - Logic Circuit Description
    - Circuit Schematic Diagrams

- **Machine Interpretation**
  - Hardware Architecture Description
    - e.g., block diagrams

---

**Levels of Representation/Interpretation**

- **Compiler**
  - \(v[k+1] = v[k] + \text{temp} \Rightarrow \text{temp} = v[k]\)
  - \(v[k+1] = v[k] + 1\)

- **Assembler**
  - \(lw \ t0, 0(\text{Rs})\)
  - \(lw \ t1, 4(\text{Rs})\)
  - \(sw \ t1, 0(\text{Rs})\)
  - \(sw \ t0, 4(\text{Rs})\)

- **Any can be represented as a number, i.e., data or instruction**

---

**Agenda**

- Overview of Control Signals
- Administrivia
- Control Implementation
- Break
- Pipelining Intro

---

**Processor Design Process**

- Five steps to design a processor:
  - Step 1: Analyze instruction set to determine datapath requirements
  - Step 2: Select set of datapath components & establish clocking methodology
  - Step 3: Assemble datapath components to meet the requirements
  - Step 4: Analyze implementation of each instruction to determine setting of control signals that implements the register transfer
  - Step 5: Assemble the control logic

---

**A Single Cycle Datapath**

- We have everything but the **values of control signals**

---

**Appendix**

- **Assembly Language Program (e.g., MIPS)**
  - \(lw \ t1, 4(\text{Rs})\)
  - \(sw \ t0, 4(\text{Rs})\)

---
Values of the Control Signals

- \( n_{PC\_sel} \): 
  - "+4" if \( n = \text{next} \) then \( 0 \Rightarrow PC \leftarrow PC + 4 \)
  - "br" if \( n = \text{next} \) then \( 1 \Rightarrow PC \leftarrow PC + 4 + \{\text{SignExt}(\text{imm16}), 00\} \)

- ExtOp: 
  - zero, sign

- ALUsrc: 
  - 0 \( \Rightarrow \text{reg} \)
  - 1 \( \Rightarrow \text{imm} \)

- ALUctr: 
  - ADD, SUB, OR

- MemWr: 
  - 1 \( \Rightarrow \text{write memory} \)

- MemtoReg: 
  - 0 \( \Rightarrow \text{alu} \)
  - 1 \( \Rightarrow \text{mem} \)

- RegDst: 
  - 0 \( \Rightarrow \text{rt} \)
  - 1 \( \Rightarrow \text{rd} \)

- RegWr: 
  - 1 \( \Rightarrow \text{write register} \)

RTL: The Add Instruction

- 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

Single Cycle Datapath during Add

- \( R[rd] = R[rs] + R[rt] \)

Instruction Fetch Unit for Add

- PC = PC + 4
  - Same for all instructions except: Branch and Jump

Single Cycle Datapath during Or Immediate

- \( R[rt] = R[rs] \text{ OR ZeroExt}[\text{imm16}] \)
Lecture #18

**Single Cycle Datapath during Or Immediate**

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

**Single Cycle Datapath during Load**

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

**Single Cycle Datapath during Branch**

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

**Instruction Fetch Unit at the End of Branch**

- If \( [Zero == 1] \) then PC = PC + 4 + SignExt[imm16]*4; else PC = PC + 4
Summary: Datapath’s Control Signals

- **ExtOp:**
  - “zero”, “sign”
- **ALUsrc:**
  - 0  regB; 1  immed
- **ALUctr:**
  - “ADD”, “SUB”, “OR”
- **MemWr:**
  - 1  write memory
- **MemtoReg:**
  - 0  ALU; 1  Mem
- **RegDst:**
  - 0  “rt”; 1  “rd”
- **RegWr:**
  - 1  write register

Processor Design Process

- **Five steps to design a processor:**
  - **Step 1:** Analyze instruction set to determine datapath requirements
  - **Step 2:** Select set of datapath components & establish clocking methodology
  - **Step 3:** Assemble datapath components to meet the requirements
  - **Step 4:** Analyze implementation of each instruction to determine setting of control signals that implements the register transfer
  - **Step 5:** Assemble the control logic

Given Datapath: RTL → Control

- Instruction set:
  - **add:** R[rd] ← R[rs] + R[rt]; PC ← PC + 4
  - ALUsrc=RegB, ALUctr="ADD", RegDest=rd, RegWr, nPC_sel="+4"
  - **sub:** R[rd] ← R[rs] – R[rt]; PC ← PC + 4
  - ALUsrc=RegB, ALUctr="SUB", RegDest=rd, RegWr, nPC_sel="+4"
  - **ori:** R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4
  - ALUsrc=Im, ExtOp="Z", ALUctr="OR", RegDst=rt, RegWr, nPC_sel="+4"
  - **lw:** R[rt] ← MEM[R[rs] + sign_ext(Imm16)]; PC ← PC + 4
  - ALUsrc=Im, ExtOp="sn", ALUctr="ADD", RegDest=rt, RegWr, nPC_sel="+4"
  - **sw:** MEM[R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4
  - ALUsrc=Im, ExtOp="sn", ALUctr="ADD", MemWrite=rd, nPC_sel="+4"
  - **beq:** if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16) || 00
  - else PC ← PC + 4
  - nPC_sel = “br”, ALUctr = “SUB"

Administrivia

- **HW3 Due Wednesday at midnight**
  - Slides at end of July 12 lecture contain useful info.
  - Project 2 Extra Credit: If you have time after finishing Part 2, try to make your program as fast as possible! Top submissions in the class get lots of extra credit, substantial improvement will get some.
- **Lab 12 cancelled!**
  - Replaced with free study session where you can catch up on labs / work on project 2.
  - The TA’s will still be there.

cs61c in the News

- Use power dissipation from servers as source of heating?
- So-called “Data Furnaces.” Sell the heat to consumers.
- “Additionally, such a setup would also provide lower network latency as the storage and computation systems can be located closer to areas of high population density and therefore those using them.”
### Summary of the Control Signals (2/2)

<table>
<thead>
<tr>
<th>ExtOp</th>
<th>nPCsel</th>
<th>MemWrite</th>
<th>RegWrite</th>
<th>MemToReg</th>
<th>ALUSrc</th>
<th>RegDst</th>
<th>Add, sub</th>
<th>ori, lw, sw, beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Agenda
- Overview of Control Signals
- Administrivia
- Control Implementation
- Break
- Pipelining Intro
### Agenda

- Overview of Control Signals
- Administrivia
- Control Implementation
- Break
- Pipelining Intro

### Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td></td>
<td>100ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>

- What can we do to improve clock rate?
- Will this improve performance as well?
  Want increased clock rate to mean faster programs

### Pipeline Analogy: Doing Laundry

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - “Folder” takes 30 minutes
  - “Stasher” takes 30 minutes to put clothes into drawers

### Sequential Laundry

- Sequential laundry takes 8 hours for 4 loads

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 PM</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>1 AM</td>
</tr>
</tbody>
</table>

### Pipelined Laundry

- Pipelined laundry takes 3.5 hours for 4 loads!

### Pipelining Lessons (1/2)

- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Time to fill pipeline and time to drain it reduces speedup: 2.3X v. 4X in this example
Pipelining Lessons (2/2)

• Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
• Pipeline rate limited by slowest pipeline stage
• Unbalanced lengths of pipe stages reduces speedup

Recall: Steps in Executing MIPS
1) **IF**: Instruction Fetch, Increment PC
2) **ID**: Instruction Decode, Read Registers
3) **EX**:
   - Mem-ref: Calculate Address
   - Arith-log: Perform Operation
4) **Mem**:
   - Load: Read Data from Memory
   - Store: Write Data to Memory
5) **WB**: Write Data Back to Register

Redrawn Single-Cycle Datapath

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction memory</th>
<th>rd</th>
<th>rt</th>
<th>imm</th>
<th>ALU</th>
<th>Data memory</th>
</tr>
</thead>
</table>

Pipelined Datapath

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction memory</th>
<th>rt</th>
<th>rd</th>
<th>imm</th>
<th>ALU</th>
<th>Data memory</th>
</tr>
</thead>
</table>

• Add registers between stages
  - Hold information produced in previous cycle

More Detailed Pipeline

IF for Load, Store, ...

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Write Back
Every instruction must take same number of steps, also called pipeline stages, so some will go idle sometimes.
Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- What is pipelined clock rate?
  - Compare pipelined datapath with single-cycle datapath

### Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions, \( T_{\text{pipelined}} \) = Time between instructions, \( T_{\text{nonpipelined}} \) / Number of stages
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

### Instruction Level Parallelism (ILP)

- ILP – e.g., Pipelined Instruction Execution
  - 5 stage pipeline => 5 instructions executing simultaneously, one at each pipeline stage
And in Conclusion, ...

• Can determine control signal values for each instruction.
• Control implementation: Just combinational logic (at least for a single cycle CPU...)
• Pipelining
  – Reduce critical path by inserting registers between each stage
  – N-stage pipeline => N operations done in parallel!

Bonus slides

• Note: You are still responsible for knowing the material covered in these slides.
• The slides will appear in the order they would have in the normal presentation

**Bonus**
How do we modify this to account for jumps?

Instruction Fetch Unit at the End of Jump

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

**Query**
- Can Zero still get asserted?
- Does nPC_sel need to be 0?
- If not, what?