Last Time

• Moving up to the microarchitecture level
  – What is a datapath?
  – How is the datapath constructed?
  – What is going on in my CPU?

Today

• Pipelining
  – Can we perform computations more efficiently than CPI = 1.0?
  – What complications arise from trying to do this?
Pipelining

- **Pipelining** – An implementation technique in which multiple instructions are overlapped in execution.
- Without pipelining:
  - 8 hours
- With pipelining:
  - 3.5 hours
- Pipelining is possible as long as we have separate resources for each stage.
Pipelining

• Benefits
  – Improves overall throughput through the system

• Drawbacks
  – Doesn’t reduce the time to fully compute a single instruction
  – Smaller instructions need to be timed to match larger instructions (it must take as long to wash, dry, fold, and put away a single sock as it does to run a complete load)
MIPS Pipelining

1. **IF**: Fetch instruction from memory.
2. **ID**: Read registers while decoding the instruction.
3. **EX**: Execute the operation or calculate the address.
4. **MEM**: Read/write from data memory.
5. **WB**: Write the result into a register.
MIPS Pipelining Performance

- Assume it takes 100ps to read/write to a register, and 200ps for the other stages

<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>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>

- Let’s compare the performance of pipelining vs. single-cycle datapath
MIPS Pipelining Performance

![Diagram of MIPS Pipelining Performance](image)
MIPS Pipelining Performance

• Single cycle = 2400ps
• Pipelined = 1300ps
• So our improvement is $2400/1300 = 1.85x$, right?
• What if we increase to a million instructions?
  – Single cycle = $1,000,000 \times 800ps = 800,000,000ps$
  – Pipelined = $1,000,000 \times 200ps \approx 200,000,000ps$
  – Improvement = $800,000,000/200,000,000 = 4x$
Pipeline Hazards

- **Structural Hazard** – A planned instruction cannot execute in the proper clock cycle because the hardware doesn’t support the combination of instructions set to execute.
  - MIPS is designed to be pipelined, so this isn’t a huge concern.
  - If we had a combined instruction & data memory segment, that would be a different story.
Pipeline Hazards

- **Data Hazard** – A planned instruction cannot execute because data that is needed to execute the instruction is not yet available.
  - Comes from the dependence of one instruction on the result of an earlier one still in the pipeline:
    
    \[
    \begin{align*}
    \text{add} & \quad \$s0, \quad \$t0, \quad \$t1 \\
    \text{sub} & \quad \$t2, \quad \$s0, \quad \$t3
    \end{align*}
    \]
  - **Solution 1**: Stall; sacrifice a few cycles
  - **Solution 2**: Add an extra structure to the datapath to retrieve $s0$ as soon as the ALU finishes the add step (called **forwarding** or **bypassing**)
Forwarding/Bypassing

• Use result when it is computed
  – Don’t wait for it to be stored in a register
  – Requires extra connections in the datapath
Forwarding/Bypassing

• Can’t always avoid stalls by forwarding
  – If instruction n-1 is a load, we need to wait an extra cycle – you can’t forward back in time
Code Scheduling to Avoid Stalls

• Reorder code to avoid use of load result in the next instruction

• C code for A = B + E; C = B + F;

```
C code for A = B + E; C = B + F;
```

```
C code for A = B + E; C = B + F;
```
Pipeline Hazards

- **Control Hazard** – A planned instruction cannot execute because the instruction that was fetched is not the one needed (also called *branch hazard*).
  - We don’t know which branch to follow for sure until the branch command is executed.
  - **Solution 1:** Stall.
  - **Solution 2:** Jump ahead.
  - **Solution 3:** Run instructions from each branch.
  - **Solution 4:** Just guess (*branch prediction*).
Branch Prediction

• **Static Prediction**
  – Based on typical branch behavior
  – Example: loop and if-statement branches
    • Predict backward branches are taken
    • Predict forward branches are not taken

• **Dynamic Prediction**
  – Hardware measures actual branch behavior
    • Record recent history of each branch
  – Assume future behavior will continue the trend
    • When wrong, stall while re-fetching, and update history
MIPS Pipelined Datapath

Right-to-left flow leads to hazards
Place Registers to Save Data Between Pipelined Stages
Let’s Look at a Load Instruction
Let’s Look at a Load Instruction
Let’s Look at a Load Instruction
Let’s Look at a Load Instruction
Let’s Look at a Load Instruction
Let’s Look at a Load Instruction
Multi-Cycle Pipeline Diagram
Single-Cycle Pipeline Diagram
Any Questions?