Instructor: Senior Lecturer SOE Dan Garcia

CS 61C: Great Ideas in Computer Architecture

Pipelining Hazards

Graphical Pipeline Diagrams

• Use datapath figure below to represent pipeline:

Graphical Pipeline Representation

• RegFile: left half is write, right half is read

Pipelining Performance (1/3)

• Use $T_c$ (“time between completion of instructions”) to measure speedup
  – $T_{c,\text{pipelined}} \geq \frac{T_{c,\text{single-cycle}}}{\text{Number of stages}}$
  – Equality only achieved if stages are balanced (i.e. take the same amount of time)
  – If not balanced, speedup is reduced
  – Speedup due to increased throughput
    – Latency for each instruction does not decrease

Pipelining Performance (2/3)

• Assume time for stages is
  – 100ps for register read or write
  – 200ps for 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>100 ps</td>
<td>800ps</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>700ps</td>
<td></td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>600ps</td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>500ps</td>
<td></td>
</tr>
</tbody>
</table>

• What is pipelined clock rate?
  – Compare pipelined datapath with single-cycle datapath

Pipelining Performance (3/3)

• Single-cycle $T_c = 800$ ps
  – $T_{c,\text{single-cycle}}$

• Pipelined $T_c = 200$ ps
  – $T_{c,\text{pipelined}}$
Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle.

1) **Structural hazard**
   - A required resource is busy (e.g. needed in multiple stages)
2) **Data hazard**
   - Data dependency between instructions
   - Need to wait for previous instruction to complete its data read/write
3) **Control hazard**
   - Flow of execution depends on previous instruction

1. **Structural Hazards**
   - Conflict for use of a resource
   - MIPS pipeline with a single memory?
     - Load/Store requires memory access for data
     - Instruction fetch would have to *stall* for that cycle
     - Causes a pipeline "bubble"
   - Hence, pipelined datapaths require separate instruction/data memories
     - Separate L1 I$ and L1 D$ take care of this

2. **Data Hazards**

   - **Consider the following sequence of instructions:**
     - `add $t0, $t1, $t2`
     - `sub $t4, $t0, $t3`
     - `and $t5, $t0, $t6`
     - `or $t7, $t0, $t8`
     - `xor $t9, $t0, $t10`
2. Data Hazards (2/2)

- Data-flow backwards in time are hazards

Data Hazard Solution: Forwarding

- Forward result as soon as it is available
  - OK that it’s not stored in RegFile yet

Datapath for Forwarding (1/2)

- What changes need to be made here?

Datapath for Forwarding (2/2)

- Handled by forwarding unit

Data Hazard: Loads (1/4)

- Recall: Dataflow backwards in time are hazards

Data Hazard: Loads (2/4)

- Hardware stalls pipeline
  - Called “hardware interlock”
Data Hazard: Loads (3/4)

- Stall is equivalent to `nop`

- Slot after a load is called a **load delay slot**
  - If that instruction uses the result of the load, then the hardware interlock will stall it for one cycle
  - Letting the hardware stall the instruction in the delay slot is equivalent to putting a `nop` in the slot (except the latter uses more code space)

**Idea**: Let the compiler put an unrelated instruction in that slot \( \rightarrow \) no stall!

Code Scheduling to Avoid Stalls

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

- MIPS code for \( D=A+B; \ E=A+C; \)

```plaintext
lb $t1, 0($t0)  # Method 1:
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
```

- **Method 2:**

```plaintext
lw $t1, 0($t0)  # Method 2:
lw $t2, 4($t0)
lw $t4, 8($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
```

Summary

- Hazards reduce effectiveness of pipelining
  - Cause stalls/bubbles
- Structural Hazards
  - Conflict in use of datapath component
- Data Hazards
  - Need to wait for result of a previous instruction
- Control Hazards
  - Address of next instruction uncertain/unknown
  - More to come next lecture!

Branch Stall

- When is comparison result available?

<table>
<thead>
<tr>
<th>Instruction Order</th>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr 1</td>
<td>15 cycles</td>
</tr>
<tr>
<td>Instr 2</td>
<td>15 cycles</td>
</tr>
<tr>
<td>Instr 3</td>
<td>15 cycles</td>
</tr>
<tr>
<td>Instr 4</td>
<td>15 cycles</td>
</tr>
</tbody>
</table>

- More to come next lecture!