CS 61C: Great Ideas in Computer Architecture

**Pipelining Hazards**

Guest Lecturer: Justin Hsia

---

### Review of Last Lecture

- Implementing controller for your datapath
  - Take decoded signals from instruction and generate control signals
  - Use “AND” and “OR” Logic scheme
- Pipelining improves performance by exploiting Instruction Level Parallelism
  - 5-stage pipeline for MIPS: IF, ID, EX, MEM, WB
  - Executes multiple instructions in parallel
  - What can go wrong???

---

### Agenda

- **Pipelining Performance**
- **Structural Hazards**
- **Administrivia**
- **Data Hazards**
  - Forwarding
  - Load Delay Slot
- **Control Hazards**

---

### Pipelined Execution Representation

**Time**

<table>
<thead>
<tr>
<th>Instruction Unit(s)</th>
<th>Functional Unit(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Core</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory</td>
</tr>
<tr>
<td>Input/Output</td>
<td></td>
</tr>
</tbody>
</table>

**Smart Phone**

Leverage Parallelism & Achieve High Performance

- **Software**
  - Parallel Requests
    - Assigned to computer
    - e.g. search “Garcia”
  - Parallel Threads
    - Assigned to core
    - e.g. backup, ads
- **Hardware**
  - Parallel instructions:
    - > 1 instruction @ one time
    - e.g. 5 pipelined instructions
  - Parallel Data:
    - > 1 data item @ one time
    - e.g. add of 4 pairs of words
  - Hardware descriptions:
    - All gates functioning in parallel at same time

---

### Pipelined Datapath

**IF**

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

- Every instruction must take same number of steps, so some stages will idle
  - e.g. MEM stage for any arithmetic instruction
Graphical Pipeline Diagrams

Graphical Pipeline Representation

Pipelining Performance (1/3)

- Use $T_c$ ("time between completion of instructions") to measure speedup
  - $T_c,\text{pipeline} \geq \frac{T_c,\text{single-cycle}}{\text{Number of stages}}$
  - 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>Instruction</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op access</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>600ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>200ps</td>
<td>700ps</td>
<td>100ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>100ps</td>
<td>600ps</td>
<td>500ps</td>
</tr>
</tbody>
</table>

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

Pipelining Performance (3/3)

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
**Agenda**

- Pipelining Performance
- Structural Hazards
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards

---

**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

---

**Structural Hazard #1: Single Memory**

- Trying to read same memory twice in same clock cycle

---

**Structural Hazard #2: Registers (1/2)**

- Can we read and write to registers simultaneously?

---

**Structural Hazard #2: Registers (2/2)**

- Two different solutions have been used:
  1) Split RegFile access in two: Write during 1st half and Read during 2nd half of each clock cycle
     - Possible because RegFile access is **VERY** fast (takes less than half the time of ALU stage)
  2) Build RegFile with independent read and write ports
- **Conclusion**: Read and Write to registers during same clock cycle is okay

---

**Agenda**

- Pipelining Performance
- Structural Hazards
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards
Administrivia

- Project 2: Performance Optimization
  - Part 1 due Sunday (4/14)
  - Part 2 released by Sunday night, due 4/21
  - Built-in performance competition for Part 2!
- Never too early to start looking at past exams!
- Dan has moved his OH because of traveling
  - See Piazza post @1492

Agenda

- Pipelining Performance
- Structural Hazards
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards

2. Data Hazards (1/2)

- 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

  \[
  \text{l}w \ $t0, 0($t1) \\
  \text{sub} \ $t3, $t0, $t2 \\
  \text{sub} \ $t5, $t0, $t4 \\
  \text{or} \ $t7,$t0,$t6
  \]

- Can’t solve all cases with forwarding
  - Must stall instruction dependent on load, then forward (more hardware)

Data Hazard: Loads (2/4)

- Hardware stalls pipeline
  - Called “hardware interlock”

  \[
  \text{l}w \ $t0, 0($t1) \\
  \text{sub} \ $t3, $t0, $t2 \\
  \text{and} \ $t5,$t0,$t4 \\
  \text{or} \ $t7,$t0,$t6
  \]

  Schematically, this is what we want, but in reality stalls done “horizontally”

Data Hazard: Loads (3/4)

- Stall is equivalent to nop

  \[
  \text{l}w \ $t0, 0($t1) \\
  \text{nop} \\
  \text{sub} \ $t3, $t0, $t2 \\
  \text{and} \ $t5,$t0,$t4 \\
  \text{or} \ $t7,$t0,$t6
  \]

- How to stall just part of pipeline?

Data Hazard: Loads (4/4)

- 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 \textit{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!

  \[
  \text{Method 1:} \\
  \text{l}w \ $t1, 0($t0) \\
  \text{l}w \ $t2, 0($t0) \\
  \text{add} \ $t3, $t2, $t1 \\
  \text{sw} \ $t3, 12($t0) \\
  \text{l}w \ $t4, 0($t0) \\
  \text{add} \ $t5, $t4, $t1 \\
  \text{sw} \ $t5, 16($t0)
  \]

  \[
  \text{Method 2:} \\
  \text{l}w \ $t1, 0($t0) \\
  \text{l}w \ $t2, 0($t0) \\
  \text{add} \ $t3, $t2, $t1 \\
  \text{sw} \ $t3, 12($t0) \\
  \text{l}w \ $t4, 0($t0) \\
  \text{add} \ $t5, $t4, $t1 \\
  \text{sw} \ $t5, 16($t0)
  \]

  \textit{13 cycles} \quad \textit{11 cycles}
Agenda

• More Pipelining
• Structural Hazards
• Adminstrivia
• Data Hazards
  – Forwarding
  – Load Delay Slot
• Control Hazards

3. Control Hazards

• Branch (beq, bne) determines flow of control
  – Fetching next instruction depends on branch outcome
  – Pipeline can’t always fetch correct instruction
    • Still working on ID stage of branch

• Simple Solution: Stall on every branch until we have the new PC value
  – How long must we stall?

Branch Stall

• When is comparison result available?

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!

Question: For each code sequences below, choose one of the statements below:

1: lw $t0,0($t0)
add $t1,$t0,$t0
2: add $t1,$t0,$t0
addi $t2,$t0,5
addi $t4,$t1,5
3: addi $t1,$t0,1
addi $t2,$t0,2
addi $t3,$t0,2
addi $t3,$t0,4
add $t5,$t1,5

A) No stalls as is
B) No stalls with forwarding
C) Must stall

Code Sequence 1
Code Sequence 2

Time (clock cycles)

No stalls with forwarding

Code Sequence 3

Time (clock cycles)

No stalls as is

7/25/2012 - Lecture #22