PIPELINING: HAZARDS

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Homework 1 is due on Sept. 12th @11:59PM
  - (Late Submission = NO submission)

- This lecture
  - Impacts of pipelining on performance
  - The MIPS five-stage pipeline
  - Pipeline hazards
    - Structural hazards
    - Data hazards
Pipelining Technique

- Improving throughput at the expense of latency
  - Delay: $D = T + n\delta$
  - Throughput: $IPS = \frac{n}{T + n\delta}$

Combinational Logic
Critical Path Delay = 30
Pipelining Technique

- Improving throughput at the expense of latency

  - **Delay:** $D = T + n\delta$

  - **Throughput:** $IPS = \frac{n}{T + n\delta}$

<table>
<thead>
<tr>
<th>Combinational Logic</th>
<th>Critical Path Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td>Comb. Logic</td>
<td>10</td>
</tr>
<tr>
<td>Comb. Logic</td>
<td>10</td>
</tr>
<tr>
<td>Comb. Logic</td>
<td>10</td>
</tr>
</tbody>
</table>

$D = \quad IPS = \quad$

$D = \quad IPS = \quad$

$D = \quad IPS = \quad$
Pipelining Technique

- Improving throughput at the expense of latency
  - Delay: $D = T + n \delta$
  - Throughput: $IPS = n / (T + n \delta)$
Pipelining Latency vs. Throughput

- Theoretical delay and throughput models for perfect pipelining
Theoretical delay and throughput models for perfect pipelining
Five Stage MIPS Pipeline
Simple Five Stage Pipeline

- A pipelined load-store architecture that processes up to one instruction per cycle
Instruction Fetch

- Read an instruction from memory (I-Memory)
  - Use the program counter (PC) to index into the I-Memory
  - Compute NPC by incrementing current PC
    - What about branches?

- Update pipeline registers
  - Write the instruction into the pipeline registers
Instruction Fetch

NPC = PC + 4

Why increment by 4?
Instruction Fetch

Critical Path = Max{P1, P2, P3}

NPC = PC + 4

Why increment by 4?

Branch Target

Memory

PC

NPC

Instruction

Register

Pipeline

Why increment by 4?

NPC = PC + 4

Critical Path = Max{P1, P2, P3}
Instruction Decode

- Generate control signals for the opcode bits

- Read source operands from the register file (RF)
  - Use the specifiers for indexing RF
    - How many read ports are required?

- Update pipeline registers
  - Send the operand and immediate values to next stage
  - Pass control signals and NPC to next stage
Instruction Decode

Pipeline Register → Instruction → Register File → decode → ctrl → NPC → Pipeline Register

target
Execute Stage

- Perform ALU operation
  - Compute the result of ALU
    - Operation type: control signals
    - First operand: contents of a register
    - Second operand: either a register or the immediate value
  - Compute branch target
    - Target = NPC + immediate

- Update pipeline registers
  - Control signals, branch target, ALU results, and destination
Execute Stage
Memory Access

- Access data memory
  - Load/store address: ALU outcome
  - Control signals determine read or write access

- Update pipeline registers
  - ALU results from execute
  - Loaded data from D-Memory
  - Destination register
Memory Access
Register Write Back

- Update register file
  - Control signals determine if a register write is needed
  - Only one write port is required
    - Write the ALU result to the destination register, or
    - Write the loaded data into the register file
Five Stage Pipeline

- Ideal pipeline: IPC=1

  - Is there enough resources to keep the pipeline stages busy all the time?
Pipeline Hazards
Pipeline Hazards

- **Structural hazards:** multiple instructions compete for the same resource

- **Data hazards:** a dependent instruction cannot proceed because it needs a value that hasn’t been produced

- **Control hazards:** the next instruction cannot be fetched because the outcome of an earlier branch is unknown
1. Unified memory for instruction and data

- $R1 \leftarrow \text{Mem}[R2]$
- $R3 \leftarrow \text{Mem}[R20]$
- $R6 \leftarrow R4 - R5$
- $R7 \leftarrow R1 + R0$
Structural Hazards

1. Unified memory for instruction and data

R1 ← Mem[R2]
R3 ← Mem[R20]
R6 ← R4-R5
R7 ← R1+R0

Separate inst. and data memories.
Structural Hazards

1. Unified memory for instruction and data
2. Register file with shared read/write access ports

R1 ← Mem[R2]
R3 ← Mem[R20]
R6 ← R4-R5
R7 ← R1+R0
Structural Hazards

- 1. Unified memory for instruction and data
- 2. Register file with shared read/write access ports

Register access in half cycles.
True dependence: read-after-write (RAW)

- Consumer has to wait for producer

Loading data from memory.

R1 ← Mem[R2]

R3 ← R1 + R0

R4 ← R1 - R3
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Loaded data will be available two cycles later.

R1 ← Mem[R2]
R3 ← R1 + R0
R4 ← R1 - R3
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Inserting two bubbles.

1. R1 ← Mem[R2]
2. Nothing
3. Nothing
4. R3 ← R1 + R0
5. R4 ← R1 - R3
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Inserting single bubble + RF bypassing.

R1 ← Mem[R2]

Nothing

R3 ← R1 + R0

R4 ← R1 - R3

Load delay slot. SW vs. HW management?
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Using the result of an ALU instruction.

R1 ← R2 + R3
R5 ← R1 + R0
R3 ← R1 + R0
R4 ← R1 - R6
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Using the result of an ALU instruction.

Forwarding ALU result.
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
  - Write must wait for earlier read

\[ R1 \leftarrow R2 + R1 \]
\[ R2 \leftarrow R8 + R9 \]
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
  - Write must wait for earlier read

R1 ← R2+R1

R2 ← R8+R9

No WAR hazards in 5-stage pipeline!
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)
  - Old writes must not overwrite the younger write

R1 ← R2 + R3

R1 ← R8 + R9
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)

- Old writes must not overwrite the younger write

No WAW hazards in 5-stage pipeline!
Data Hazards

- Forwarding with additional hardware
Data Hazards

- How to detect and resolve data hazards

  - Show all of the data hazards in the code below

```plaintext
R1 ← Mem[R2]
R2 ← R1 + R0
R1 ← R1 - R2
Mem[R3] ← R2
```
How to detect and resolve data hazards

Show all of the data hazards in the code below

```
R1 ← Mem[R2]
R2 ← R1 + R0
R1 ← R1 - R2
Mem[R3] ← R2
```