ECE 4750 Computer Architecture
Topic 3: Pipelining
Structural & Data Hazards

Christopher Batten
School of Electrical and Computer Engineering
Cornell University

http://www.csl.cornell.edu/courses/ece4750
slide revision: 2012-09-05-17-41
Iron Law of Processor Performance

\[
\text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycles}}
\]

<table>
<thead>
<tr>
<th>Microarchitecture</th>
<th>CPI</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic 2 → Microcoded</td>
<td>&gt;1</td>
<td>short</td>
</tr>
<tr>
<td>topic 3 → Single-Cycle Unpipelined</td>
<td>1</td>
<td>long</td>
</tr>
<tr>
<td>topic 3 → Multi-Cycle Unpipelined</td>
<td>&gt;1</td>
<td>short</td>
</tr>
<tr>
<td>topics 4–6 → Pipelined</td>
<td>≈1</td>
<td>short</td>
</tr>
</tbody>
</table>
CPI Of Various Microarchitectures

**Microcoded**
- **CPI = 7.33**
- Inst 1: 7 cycles
- Inst 2: 5 cycles
- Inst 3: 10 cycles

**Single-Cycle Unpipelined**
- **CPI = 1**
- Inst 1: 1 cycle
- Inst 2: 1 cycle
- Inst 3: 1 cycle

**Multi-Cycle Unpipelined**
- **CPI = 4.33**
- Inst 1: 5 cycles
- Inst 2: 3 cycles
- Inst 3: 5 cycles

**Pipelined**
- **CPI = 1**
- Inst 1 (5 cycles)
- Inst 2 (5 cycles)
- Inst 3 (5 cycles)
Agenda

Pipelining Basics

Structural Hazards

Data Hazards
**An Ideal Pipeline**

- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- Scheduling of a transaction entering pipeline is not affected by transactions in other stages
- These conditions generally hold for industry assembly lines, but instructions depend on each other causing various hazards
Pipelining Basics

Pipelined MIPS Processor: Start With Multi-Cycle Unpipelined Processor

Clock period is reduced by dividing the execution of an instruction into multiple cycles

\[ t_c < \max( t_{ifetch}, t_{rf}, t_{ALU}, t_{dmem}, t_{rfwr} ) \]
Pipelined MIPS Processor: Add Pipeline Registers to Control Unit

As instruction goes down the pipeline, fewer control bits are needed.
Pipelining Technology Assumptions

- Small amount of very fast memory (caches), backed by large, slower memory
- Fast ALU (at least for integers)
- Multiported register files (slower!)

Thus, the following timing assumption is reasonable

\[ t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW} \]

A 5-stage pipeline will be the focus of our detailed design, although real commercial designs usually have many more stages.
We need to be able to show multiple simultaneous transactions in both space and time.
Pipeline Diagrams: Transactions vs. Time

- Pipelining Basics
- Structural Hazards
- Data Hazards
Pipeline Diagrams: Space vs. Time

- **Pipelining Basics**
- **Structural Hazards**
- **Data Hazards**

---

**Resources**

- **IF**
  - time: t0, t1, t2, t3, t4, t5, t6, t7, ...
  - Resources: I1, I2, I3, I4, I5
- **ID**
- **EX**
- **MA**
- **WB**

---

**Diagram Description**

- **Fetch Phase**
  - PC -> Addr
  - Addr -> Inst. Memory

- **Decode & Reg-fetch Phase**
  - IF: I1, I2, I3, I4, I5
  - ID: I1, I2, I3, I4, I5
  - Ex: I1, I2, I3, I4, I5

- **Execute Phase**
  - ALU

- **Memory Phase**
  - Data Memory
  - Data Memory: wdata

- **Write-back Phase**
  - WB
Instructions Interact With Each Other in Pipeline

- **Structural Hazard** – An instruction in the pipeline needs a resource being used by another instruction in the pipeline.

- **Data Hazard** – An instruction depends on a data value produced by an earlier instruction.

- **Control Hazard** – Whether or not an instruction should be executed depends on a control decision made by an earlier instruction.
Pipelining Basics

Structural Hazards

Data Hazards
Overview Structural Hazards

- Structural hazards occur when two instructions need the same hardware resource at the same time.

- Approaches to resolving structural hazards:
  - **Schedule** – Programmer explicitly avoids scheduling instructions that would create structural hazards.
  - **Stall** – Hardware includes control logic that stalls until earlier instruction is no longer using contended resource.
  - **Duplicate** – Add more hardware to design so that each instruction can access to independent resources at the same time.

- Simple 5-stage MIPS pipeline has no structural hazards specifically because ISA was designed that way.
Example Structural Hazard: Unified Memory

Pipeline diagram on board
Example Structural Hazard: 2-Cycle Data Memory

Pipeline diagram on board
Agenda

Pipelining Basics

Structural Hazards

Data Hazards
Overview of Data Hazards

- Data hazards occur when one instruction depends on a data value produced by an preceding instruction still in the pipeline

- Approaches to resolving data hazards
  - **Schedule** – Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall** – Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass** – Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate** – Guess that there is not a problem, if incorrect kill speculative instruction and restart
Example Data Hazard

\[ r_4 \leftarrow r_1 \ldots \]
\[ r_1 \leftarrow \ldots \]

\[ r_1 \leftarrow r_0 + 10 \]
\[ r_4 \leftarrow r_1 + 17 \]

\[ \ldots \]

Pipeline diagram on board
Feedback to Resolve Hazards

Later stages provide dependence information to earlier stages which can stall (or kill) following instructions.

Controlling a pipeline in this manner works provided the instruction at stage i+1 can complete without any interaction from instructions in stages 1 to i (otherwise deadlock).
Resolving Data Hazards with Stalls

Stall Condition

... r1 ← r0 + 10
r4 ← r1 + 17
...

Pipeline diagram on board
Stalled Stages and Pipeline Bubbles

\[ (I_1) \ r_1 \leftarrow (r_0) + 10 \hspace{1cm} IF_1 \]
\[ (I_2) \ r_4 \leftarrow (r_1) + 17 \hspace{1cm} ID_2 \]
\[ (I_3) \hspace{1cm} \]
\[ (I_4) \hspace{1cm} \]
\[ (I_5) \hspace{1cm} \]

\( time \)
\[ t_0 \hspace{1cm} t_1 \hspace{1cm} t_2 \hspace{1cm} t_3 \hspace{1cm} t_4 \hspace{1cm} t_5 \hspace{1cm} t_6 \hspace{1cm} t_7 \hspace{1cm} \ldots \]

IF: IF \[ I_1 \hspace{1cm} I_2 \hspace{1cm} I_3 \hspace{1cm} I_3 \hspace{1cm} I_3 \hspace{1cm} I_3 \hspace{1cm} I_4 \hspace{1cm} I_5 \]
ID: ID \[ I_1 \hspace{1cm} I_2 \hspace{1cm} I_2 \hspace{1cm} I_2 \hspace{1cm} I_2 \hspace{1cm} I_2 \hspace{1cm} I_2 \hspace{1cm} I_2 \]
EX: EX \[ I_1 \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} I_2 \hspace{1cm} I_3 \hspace{1cm} I_4 \hspace{1cm} I_5 \]
MA: MA \[ I_1 \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} I_2 \hspace{1cm} I_3 \hspace{1cm} I_4 \hspace{1cm} I_5 \]
WB: WB \[ I_1 \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} nop \hspace{1cm} I_2 \hspace{1cm} I_3 \hspace{1cm} I_4 \hspace{1cm} I_5 \]

\( nop \Rightarrow \) pipeline bubble

ECE 4750 T03: Pipelining – Structural & Data Hazards 22 / 35
Stall Control logic

Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions.
Stall Control logic (ignoring jumps/branches)

Always stall if rs field matches some rd?
Not every instruction writes or reads a register!
## Source and Destination Registers

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source (srcs)</th>
<th>Destination (dest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>R[rd] ← R[rs] func R[rt]</td>
<td>rs, rt rd</td>
</tr>
<tr>
<td>ALUI</td>
<td>R[rt] ← R[rs] op immediate</td>
<td>rs rt</td>
</tr>
<tr>
<td>LW</td>
<td>R[rt] ← M[ R[rs] + offset ]</td>
<td>rs rt</td>
</tr>
<tr>
<td>SW</td>
<td>M[ R[rs] + offset ] ← R[rt]</td>
<td>rs rt</td>
</tr>
<tr>
<td>BEQZ</td>
<td>if ( R[rs] == 0 ) PC ← PC+4 + offset*4</td>
<td>rs</td>
</tr>
<tr>
<td>J</td>
<td>PC ← (\text{j targ}(\text{PC,imm}))</td>
<td></td>
</tr>
<tr>
<td>JAL</td>
<td>R[31] ← PC; PC ← (\text{j targ}(\text{PC,imm}))</td>
<td>31</td>
</tr>
<tr>
<td>JR</td>
<td>PC ← R[rs]</td>
<td>rs</td>
</tr>
<tr>
<td>JALR</td>
<td>R[31] ← PC, PC ← R[rs]</td>
<td>rs 31</td>
</tr>
</tbody>
</table>
Deriving the Stall Signal

\[ C_{d_e} \]
\[ ws = \text{Case opcode} \]
\[ \text{ALU} \Rightarrow rd \]
\[ \text{ALUi, LW} \Rightarrow rt \]
\[ \text{JAL, JALR} \Rightarrow R31 \]

\[ we = \text{Case opcode} \]
\[ \text{ALU, ALUi, LW} \Rightarrow (ws \neq 0) \]
\[ \text{JAL, JALR} \Rightarrow \text{on} \]
\[ ... \Rightarrow \text{off} \]

\[ C_{re} \]
\[ re1 = \text{Case opcode} \]
\[ \text{ALU, ALUi, LW, SW, BZ, JR, JALR} \Rightarrow \text{on} \]
\[ \text{J, JAL} \Rightarrow \text{off} \]

\[ re2 = \text{Case opcode} \]
\[ \text{ALU, SW} \Rightarrow \text{on} \]
\[ ... \Rightarrow \text{off} \]

\[ C_{s_t} \]
\[ \text{stall} = ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W) \cdot re1_D + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W) \cdot re2_D \]

This is not the full story!
Is there any possible data hazard in this instruction sequence?
Data Hazards Due to Loads and Stores

- Example instruction sequence
  - $M[R[r1] + 7] \leftarrow R[r2]$
  - $R[r4] \leftarrow M[R[r3] + 5]$

- What if $R[r1]+7 == R[r3]+5$?
  - Writing and reading from the same address
  - Hazard is avoided because our memory system completes writes in a single cycle
  - More realistic memory system will require more careful handling of data hazards due to loads and stores

Pipeline diagram on board
Adding a Bypass to the Datapath

When does this bypass help?

(I_1) \( r1 ← r0 + 10 \)  |  \( r1 ← M[r0 + 10] \)  |  \text{JAL 500}  
(I_2) \( r4 ← r1 + 17 \)  |  \( r4 ← r1 + 17 \)  |  \( r4 ← r31 + 17 \)
Deriving the Bypass Signal

Each *stall or kill* introduces a bubble in the pipeline

⇒ CPI > 1

A new datapath, i.e., a *bypass*, can get the data from the output of the ALU to its input
Deriving the Bypass Signal

\[
\text{stall} = (((rs_\text{D} = ws_{\text{E}}).we_{\text{E}} + (rs_\text{D} = ws_\text{M}).we_{\text{M}} + (rs_\text{D} = ws_\text{W}).we_{\text{W}}).re_{1D} \\
\quad +((rt_\text{D} = ws_{\text{E}}).we_{\text{E}} + (rt_\text{D} = ws_\text{M}).we_{\text{M}} + (rt_\text{D} = ws_\text{W}).we_{\text{W}}).re_{2D})
\]

\[
ws = \text{Case opcode} \\
\begin{align*}
\text{ALU} & \Rightarrow \text{rd} \\
\text{ALUi, LW} & \Rightarrow \text{rt} \\
\text{JAL, JALR} & \Rightarrow \text{R31}
\end{align*}
\]

\[
\text{we} = \text{Case opcode} \\
\begin{align*}
\text{ALU, ALUi, LW} & \Rightarrow (ws \neq 0) \\
\text{JAL, JALR} & \Rightarrow \text{on} \\
... & \Rightarrow \text{off}
\end{align*}
\]

\[
\text{ASrc} = (rs_\text{D} = ws_{\text{E}}).we_{\text{E}}.re_{1D}
\]

Is this correct?

No, because only ALU and ALUI instruction can benefit from this bypass.

Split \text{we}_{\text{E}} into two components: we-bypass and we-stall.
Bypass and Stall Signals

Split $we_E$ into two components: $we$-bypass and $we$-stall

\[
\begin{align*}
we\text{-bypass}_E &= \text{Case opcode}_E \\
& \text{ALU, ALUi} \quad \Rightarrow (ws \neq 0) \\
& \quad \Rightarrow \text{off}
\end{align*}
\]

\[
\begin{align*}
we\text{-stall}_E &= \text{Case opcode}_E \\
& \text{LW} \quad \Rightarrow (ws \neq 0) \\
& \quad \Rightarrow \text{off}
\end{align*}
\]

\[
\begin{align*}
\text{ASrc} &= (rs_D = ws_E).\text{we\text{-bypass}_E} \cdot \text{re1}_D \\
\text{stall} &= ((rs_D = ws_E).\text{we\text{-stall}_E} + \\
& \quad (rs_D = ws_M).\text{we}_M + (rs_D = ws_W).\text{we}_W). \text{re1}_D \\
& +((rt_D = ws_E).\text{we}_E + (rt_D = ws_M).\text{we}_M + (rt_D = ws_W).\text{we}_W). \text{re2}_D
\end{align*}
\]
Fully Bypassed Datapath

Is there still a need for the stall signal?

\[
\text{stall} = (rs_D = ws_E). (\text{opcode}_E = \text{LW}_E). (ws_E \neq 0). \text{re1}_D \\
+ (rt_D = ws_E). (\text{opcode}_E = \text{LW}_E). (ws_E \neq 0). \text{re2}_D
\]
Summary

- **Structural Hazard** – An instruction in the pipeline needs a resource being used by another instruction in the pipeline *(this topic)*

- **Data Hazard** – An instruction depends on a data value produced by an earlier instruction *(this topic)*

- **Control Hazard** – Whether or not an instruction should be executed depends on a control decision made by an earlier instruction *(next topic)*
Some of these slides contain material developed and copyrighted by:

Arvind (MIT), Krste Asanović (MIT/UCB), Joel Emer (Intel/MIT)
James Hoe (CMU), John Kubiatowicz (UCB), David Patterson (UCB)

MIT material derived from course 6.823
UCB material derived from courses CS152 and CS252