Instruction Pipelining and Hazards

Daniel Sanchez
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Reminder: Harvard-Style Single-Cycle Datapath for MIPS
Princeton challenge

What problem arises if we use a single memory to hold instructions and data?

At least the instruction fetch and a Load (or Store) cannot be executed in the same cycle

Structural hazard
Two-State Controller
Princeton Architecture

fetch phase

execute phase

A flip-flop can be used to remember the phase
Control Logic
Princeton Architecture

old combinational logic (Harvard)

IR

opcode

zero?

ExtSel, BSrc, OpSel, WBSrc, RegDest, PCsrc1, PCsrc2

MemWrite

RegWrite

new combinational logic

1-bit Toggle FF

Fetch / Execute

S

Wen

PCen

IRen

AddrSrc
Princeton Microarchitecture
Datapath & Control for 2 cycles-per-instruction

February 25, 2019
The same (mux not shown)

Only one of the phases is active in any cycle
⇒ a lot of datapath is not in use at any given time
Princeton Microarchitecture
Overlapped execution

Can we overlap instruction fetch and execute?
Yes, unless IR contains a Load or Store

Which action should be prioritized? Execute

What do we do with Fetch? Stall it How?
When stall condition is indicated

- don’t fetch a new instruction and don’t change the PC
- insert a nop in the IR
- set the Memory Address mux to ALU (not shown)

What if IR contains a jump or branch instruction?
Need to stall on branches
Princeton Microarchitecture

When IR contains a jump or taken branch
• no "structural conflict" for the memory
• but we do not have the correct PC value in the PC
• memory cannot be used – Address Mux setting is irrelevant
• insert a nop in the IR
• insert the nextPC (branch-target) address in the PC
Pipelined Princeton Microarchitecture
## Pipelined Princeton: Control Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Stall</th>
<th>Ext Sel</th>
<th>B Src</th>
<th>Op Sel</th>
<th>Mem W</th>
<th>Reg W</th>
<th>WB Src</th>
<th>Reg Dst</th>
<th>PC Src1</th>
<th>PC Src2</th>
<th>IR Src</th>
<th>MAddr Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>no</td>
<td>*</td>
<td>Reg</td>
<td>Func</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rd</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>ALUi</td>
<td>no</td>
<td>sE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Imm</td>
<td>Op</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rt</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>ALUiu</td>
<td>no</td>
<td>uE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Imm</td>
<td>Op</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rt</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>LW</td>
<td>yes</td>
<td>sE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Imm</td>
<td>+</td>
<td>no</td>
<td>yes</td>
<td>Mem</td>
<td>rt</td>
<td>pc+4</td>
<td>pc</td>
<td>nop</td>
<td>ALU</td>
</tr>
<tr>
<td>SW</td>
<td>yes</td>
<td>sE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Imm</td>
<td>+</td>
<td>yes</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
<td>pc</td>
<td>nop</td>
<td>ALU</td>
</tr>
<tr>
<td>BEQZ&lt;sub&gt;z=1&lt;/sub&gt;</td>
<td>yes</td>
<td>sE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>*</td>
<td>0?</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>br</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>BEQZ&lt;sub&gt;z=0&lt;/sub&gt;</td>
<td>no</td>
<td>sE&lt;sub&gt;16&lt;/sub&gt;</td>
<td>*</td>
<td>0?</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>J</td>
<td>yes</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>jabs</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>JAL</td>
<td>yes</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>PC</td>
<td>R31</td>
<td>jabs</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>JR</td>
<td>yes</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>rind</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>JALR</td>
<td>yes</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>PC</td>
<td>R31</td>
<td>rind</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>NOP</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
</tbody>
</table>

**BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU**

*RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

**stall & IRSrc columns are identical**
Pipelined Princeton Architecture

Clock: \[ t_{\text{C-Princeton}} > t_{\text{RF}} + t_{\text{ALU}} + t_{\text{M}} + t_{\text{WB}} \]

CPI: \[(1 - f) + 2f \text{ cycles per instruction}\]
where \(f\) is the fraction of instructions that cause a stall

What is a likely value of \(f\)?
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
- The scheduling of an object entering the pipeline is not affected by the objects in other stages

*These conditions generally hold for industrial assembly lines.*

*But what about an instruction pipeline?*
Clock period can be reduced by dividing the execution of an instruction into multiple cycles

\[ t_C > \max \{ t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW} \} \ (= t_{DM} \text{ probably}) \]

However, CPI will increase unless instructions are pipelined
How to divide datapath into stages

Suppose memory is significantly slower than other stages. For example, suppose

\[
\begin{align*}
t_{IM} & = 10 \text{ units} \\
t_{DM} & = 10 \text{ units} \\
t_{ALU} & = 5 \text{ units} \\
t_{RF} & = 1 \text{ unit} \\
t_{RW} & = 1 \text{ unit}
\end{align*}
\]

Since the slowest stage determines the clock, it may be possible to combine some stages without any loss of performance.
Alternative Pipelining

\[ t_C > \max \{ t_{IM}, t_{RF} + t_{ALU}, t_{DM} + t_{RW} \} = t_{DM} + t_{RW} \]

\( \Rightarrow \text{increase the critical path by 10\%} \)

Write-back stage takes much less time than other stages. Suppose we combined it with the memory phase.
# Maximum Speedup by Pipelining

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Unpipelined $t_C$</th>
<th>Pipelined $t_C$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $t_{IM} = t_{DM} = 10$, $t_{ALU} = 5$, $t_{RF} = t_{RW} = 1$ 4-stage pipeline</td>
<td>27</td>
<td>10</td>
<td>2.7</td>
</tr>
<tr>
<td>2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline</td>
<td>25</td>
<td>10</td>
<td>2.5</td>
</tr>
<tr>
<td>3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline</td>
<td>25</td>
<td>5</td>
<td>5.0</td>
</tr>
</tbody>
</table>

What seems to be the message here?

*One can achieve higher speedup with more pipeline stages*
5-Stage Pipelined Execution

Instruction Flow Diagram

- **I-Fetch (IF)**
- **Decode, Reg. Fetch (ID)**
- **Execute (EX)**
- **Memory (MA)**
- **Write-Back (WB)**

**Time Steps:**
- t0, t1, t2, t3, t4, t5, t6, t7, ...

**Instructions:**
- Instruction1
- Instruction2
- Instruction3
- Instruction4
- Instruction5

**Stages:**
- IF
- ID
- EX
- MA
- WB

**Components:**
- Addr, Rdata
- PC, IR
- Imm, Ext
- ALU
- GPRs
- IR, PC
5-Stage Pipelined Execution

Resource Usage Diagram

Resources

- IF
- ID
- EX
- MA
- WB

Time

- t0
- t1
- t2
- t3
- t4
- t5
- t6
- t7
- . . .

IF

ID

EX

MA

WB

Write-Back (WB)
Pipelined Execution

ALU Instructions

Not quite correct!

We need an Instruction Reg (IR) for each stage
Pipelined MIPS Datapath
without jumps

What else is needed?

Control Points Need to Be Connected
How instructions can interact with each other in a pipeline

• An instruction in the pipeline may need a resource being used by another instruction in the pipeline → structural hazard

• An instruction may depend on a value produced by an earlier instruction
  – Dependence may be for a data calculation → data hazard
  – Dependence may be for calculating the next PC → control hazard (branches, interrupts)
Data Hazards

... r1 ← r0 + 10
r4 ← r1 + 17 ...

r1 is stale. Oops!
Resolving Data Hazards

Strategy 1: *Wait for the result to be available by freezing earlier pipeline stages* → *stall*

Strategy 2: *Route data as soon as possible after it is calculated to the earlier pipeline stage* → *bypass*

Strategy 3: *Speculate on the dependence*

*Two cases:*

- *Guessed correctly* → *do nothing*
- *Guessed incorrectly* → *kill and restart*
Resolving Data Hazards (1)

Strategy 1:

Wait for the result to be available by freezing earlier pipeline stages \(\rightarrow\) stall
Feedback to Resolve Hazards

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

• Controlling a pipeline in this manner works provided the instruction at stage $i+1$ can complete without any interference from instructions in stages 1 to $i$ (otherwise deadlocks may occur)
Resolving Data Hazards by Stalling

Stall Condition

PC ➔ 0x4
addr ➔ Add
Inst ➔ Memory

nop ➔ IR

... r1 ← r0 + 10
... r4 ← r1 + 17
...

...
Stalled Stages and Pipeline Bubbles

\[ \text{time} \]
\[
\begin{array}{ccccccccc}
  t0 & t1 & t2 & t3 & t4 & t5 & t6 & t7 & \ldots
  \\
  \text{IF} & I_1 & I_2 & I_3 & I_3 & I_3 & I_3 & I_3 & \ldots
  \\
  \text{ID} & I_1 & I_2 & I_2 & I_2 & I_2 & I_2 & I_2 & \ldots
  \\
  \text{EX} & \text{n} \text{op} & \text{n} \text{op} & \text{n} \text{op} & \text{n} \text{op} & \text{n} \text{op} & \text{n} \text{op} & \text{n} \text{op} & \ldots
  \\
  \text{MA} & I_1 & I_2 & I_2 & I_2 & I_2 & I_2 & I_2 & \ldots
  \\
  \text{WB} & I_1 & I_2 & I_2 & I_2 & I_2 & I_2 & I_2 & \ldots
  \\
\end{array}
\]

\( (I_1) \ r_1 \leftarrow (r_0) + 10 \)
\( (I_2) \ r_4 \leftarrow (r_1) + 17 \)

\text{Resource Usage}

\text{nop} \ \Rightarrow \ \text{pipeline bubble}
Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions.
Thank you!

Next lecture: Control Hazards, Bypassing and Speculation