Instruction Pipelining

Ryan Lee
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Announcements

• Lab 1 released later today
  - Designing 3 different cache models using Pin
  - Due Oct. 1

• Please view the Pin tutorial video posted on the course website (under Recitation tab)

• Contact me if you cannot get access to lab machines
Reminder: Harvard-Style Single-Cycle Datapath for MIPS

PCSrc \(\rightarrow\) br \(\rightarrow\) rind \(\rightarrow\) jabs \(\rightarrow\) pc+4

0x4 \(\rightarrow\) Add

RegWrite

MemWrite \(\rightarrow\) WBSrc

Add

Inst. Memory

OPCode \(\rightarrow\) RegDst \(\rightarrow\) ExtSel \(\rightarrow\) OpSel \(\rightarrow\) BSrc \(\rightarrow\) zero?

ALU

Control

Data Memory

rd1 \(\rightarrow\) rs1 \(\rightarrow\) rs2

addrs \(\rightarrow\) inst \(\rightarrow\) GPRs

we \(\rightarrow\) rd1 \(\rightarrow\) we

wdata \(\rightarrow\) rd2

rdaddr \(\rightarrow\) wdata

March 4, 2021

MIT 6.823 Spring 2021
Reminder: Princeton Microarchitecture
Datapath & Control for 2 cycles-per-instruction
Princeton Microarchitecture (redrawn)

Only one of the phases is active in any cycle
⇒ a lot of datapath not used at any given time
Can we overlap instruction fetch and execute?

Which action should be prioritized?

What do we do with Fetch?
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?**
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>sE₁₆</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>uE₁₆</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>sE₁₆</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>sE₁₆</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>BEQZ₂=1</td>
<td>yes</td>
<td>sE₁₆</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>BEQZ₂=0</td>
<td>no</td>
<td>sE₁₆</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; MAddrSrc = 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: \( (1 - f) + 2f \) 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

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

However, CPI will increase unless instructions are pipelined
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.
Write-back stage takes much less time than other stages. Suppose we combined it with the memory phase.
Maximum Speedup by Pipelining

Assumptions

1. \( t_{IM} = t_{DM} = 10, \)
   \( t_{ALU} = 5, \)
   \( t_{RF} = t_{RW} = 1 \)
   4-stage pipeline

2. \( t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5 \)
   4-stage pipeline

3. \( t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5 \)
   5-stage pipeline

What seems to be the message here?
5-Stage Pipelined Execution

Instruction Flow Diagram

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

**Time and Instructions**

- **Instruction 1**
  - Time: t0
  - Stage: IF
- **Instruction 2**
  - Time: t1
  - Stage: IF
- **Instruction 3**
  - Time: t2
  - Stage: ID
- **Instruction 4**
  - Time: t3
  - Stage: MA
- **Instruction 5**
  - Time: t4
  - Stage: WB

**Stages**

- Fetch (IF)
- Decode, Register Fetch (ID)
- Execute (EX)
- Memory (MA)
- Write-Back (WB)
5-Stage Pipelined Execution

Resource Usage Diagram

Resources

<table>
<thead>
<tr>
<th>time</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td>I1</td>
<td>I1</td>
<td>I1</td>
<td>I1</td>
<td>I1</td>
</tr>
<tr>
<td>t1</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
</tr>
<tr>
<td>t2</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
</tr>
<tr>
<td>t3</td>
<td>I4</td>
<td>I4</td>
<td>I4</td>
<td>I4</td>
<td>I4</td>
</tr>
<tr>
<td>t4</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
</tr>
<tr>
<td>t5</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
<td>I5</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

March 4, 2021

MIT 6.823 Spring 2021

L06-18
Pipelined Execution

ALU Instructions

Not quite correct!
Pipelined MIPS Datapath
without jumps

What else is needed?
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

r4 ← r1 ...

... r1 ← r0 + 10
r4 ← r1 + 17 ...

r1 is stale. Oops!
Resolving Data Hazards

Use strategy from Princeton Pipeline:

Wait for the result to be available by freezing earlier pipeline stages → stall
Later stages provide dependence information to earlier stages which can stall 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**

... r1 ← r0 + 10
r4 ← r1 + 17
...

March 4, 2021
Stalled Stages and Pipeline Bubbles

Resource Usage

<table>
<thead>
<tr>
<th>time</th>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₃</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₅</td>
</tr>
<tr>
<td>ID</td>
<td>I₁</td>
<td>I₁</td>
<td>I₁</td>
<td>I₂</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td>I₁</td>
<td>nop</td>
<td>nop</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
</tr>
<tr>
<td>MA</td>
<td></td>
<td></td>
<td></td>
<td>nop</td>
<td>nop</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nop</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
</tr>
</tbody>
</table>

Stalled stages

IF₁ 
ID₁ 
EX₁ MA₁ WB₁
ID₂ ID₂ ID₂
IF₃ IF₃ IF₃

(I₁) r₁ ← (r₀) + 10
(I₂) r₄ ← (r₁) + 17
(I₃)
(I₄)
(I₅)

nop ⇒ pipeline bubble

March 4, 2021

MIT 6.823 Spring 2021
Stall Control Logic

Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions.
Should we always stall if the rs field matches some rd?
Thank you!