Instruction Pipelining

Atalay Mert Ileri
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Announcements

• Lab 1 released later today
  – Designing 3 different cache models using Pin
  – Due Sept. 30

• 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
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

fetch phase

execute phase

The same (mux not shown)
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
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>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:**  \[ t_{C-Princeton} > t_{RF} + t_{ALU} + t_M + t_{WB} \]

**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?
Pipelined Datapath

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

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

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

$\Rightarrow$ increase the critical path by 10%
# 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. $t_{IM} = t_{DM} = 10$, $t_{ALU} = 5$, $t_{RF} = t_{RW} = 1$</td>
<td>27</td>
<td>10</td>
<td>2.7</td>
</tr>
<tr>
<td>2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$</td>
<td>25</td>
<td>10</td>
<td>2.5</td>
</tr>
<tr>
<td>3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$</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)**
  - Time: t0, t1, t2, t3, t4, t5, t6, t7, ...
  - Instruction 1, Instruction 2, Instruction 3, Instruction 4, Instruction 5

- **Decode, Reg. Fetch (ID)**
  - Instruction 1
  - rd1, rs1, rs2

- **Execute (EX)**
  - Instruction 2
  - wk, wn, rd2

- **Memory (MA)**
  - Instruction 3
  - addr, rdata

- **Write-Back (WB)**
  - Instruction 4
  - we, wdata

PC → Add → Inst. Memory

- **Add**
  - 0x4

- **Inst. Memory**
  - addr, rdata

- **IR**

- **Decode**
  - Imm, Ext

- **ALU**
  - Imm, Ext, Ext

- **Memory**
  - wb, wdata

- **Write-Back**
  - write-back

September 16, 2022
5-Stage Pipelined Execution

Resource Usage Diagram

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

The diagram illustrates the pipeline stages with resource usage and time intervals:

- **Resources**
  - **IF**
  - **ID**
  - **EX**
  - **MA**
  - **WB**

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

- **Instruction Flow**
  - From PC to Instruction Memory (IF)
  - From Instruction Memory to Decode and Register Fetch (ID)
  - From Decode and Register Fetch to Execute (EX)
  - From Execute to Memory (MA)
  - From Memory to Write-Back (WB)

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

The diagram also includes details such as PC, IR, GPRs, ALU, Imm Ext, and memory operations like Addr, Rdata, Wdata, and Write operations.
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

\[ r1 \leftarrow r0 + 10 \]
\[ r4 \leftarrow r1 + 17 \]

\[ r1 \text{ is stale. Oops!} \]
Resolving Data Hazards

*Use strategy from Princeton Pipeline:*

*Wait for the result to be available by freezing earlier pipeline stages → stall*
Feedback to Resolve Hazards

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

\[ \text{Stall Condition} \]

\[ \text{...} \]
\[ r1 \leftarrow r0 + 10 \]
\[ r4 \leftarrow r1 + 17 \]
\[ \text{...} \]
Stalled Stages and Pipeline Bubbles

Stalled Stages

IF   ID   EX   MA   WB
(I_1) r1 ← (r0) + 10
(I_2) r4 ← (r1) + 17
(I_3)
(I_4)
(I_5)

Resource Usage

nop ⇒ pipeline bubble

September 16, 2022
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?

not every instruction writes a register ⇒ we
not every instruction reads a register ⇒ re
Thank you!