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
Reminder: Princeton Microarchitecture
Datapath & Control for 2 cycles-per-instruction

March 4, 2021
The same (mux not shown)

Only one of the phases is active in any cycle
⇒ a lot of datapath not used 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

- IR
- RegDst
- PCSrc
- RegWrite
- BSrc
- zero?
- WBSrc
- IRSrc
- PCSrc2
- MemWrite
- pclk
- ExtSel
- OpCode
- GPRs
- rs1
- rs2
- rd1
- we
- ws
- wd
- rd2
- we
- MemWrite
- nop
- ALU
- Control
- stall?
## Pipelined Princeton: Control Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Stall</th>
<th>Ext Sel</th>
<th>B Src</th>
<th>Opsel</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>ncp</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>ncp</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>ncp</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>ncp</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>ncp</td>
<td>ALU</td>
</tr>
<tr>
<td>BEQZ_{z=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>ncp</td>
<td>ncp</td>
<td>*</td>
</tr>
<tr>
<td>BEQZ_{z=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>ncp</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>ncp</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>ncp</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>ncp</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>ncp</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>ncp</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

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

\[ \text{CPI:} \quad (1 - f) + 2f \text{ cycles per instruction} \]
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

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

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

\[ t_{IM} = 10 \text{ units} \]
\[ t_{DM} = 10 \text{ units} \]
\[ t_{ALU} = 5 \text{ units} \]
\[ t_{RF} = 1 \text{ unit} \]
\[ t_{RW} = 1 \text{ unit} \]

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

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

$\Rightarrow$ 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

### Assumptions

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

*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)

<table>
<thead>
<tr>
<th>time</th>
<th>instruction1</th>
<th>instruction2</th>
<th>instruction3</th>
<th>instruction4</th>
<th>instruction5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;2&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;3&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;4&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>t1</td>
<td>ID&lt;sub&gt;1&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;2&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;3&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;4&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>t2</td>
<td>EX&lt;sub&gt;1&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;2&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;3&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;4&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>t3</td>
<td>MA&lt;sub&gt;1&lt;/sub&gt;</td>
<td>MA&lt;sub&gt;2&lt;/sub&gt;</td>
<td>MA&lt;sub&gt;3&lt;/sub&gt;</td>
<td>MA&lt;sub&gt;4&lt;/sub&gt;</td>
<td>MA&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>t4</td>
<td>WB&lt;sub&gt;1&lt;/sub&gt;</td>
<td>WB&lt;sub&gt;2&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td>WB&lt;sub&gt;3&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td>WB&lt;sub&gt;4&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>WB&lt;sub&gt;5&lt;/sub&gt;</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>
5-Stage Pipelined Execution

Resource Usage Diagram

- **I-Fetch (IF)**
  - Time: t0, t1, t2, t3, t4, t5, t6, t7, ...
  - Resources: IF, ID, EX, MA, WB

- **Decode, Reg. Fetch (ID)**
  - Resources: IF, ID, EX, MA, WB

- **Execute (EX)**
  - Resources: IF, ID, EX, MA, WB

- **Memory (MA)**
  - Resources: IF, ID, EX, MA, WB

- **Write-Back (WB)**
  - Resources: IF, ID, EX, MA, WB
Pipelined Execution

ALU Instructions

Not quite correct!

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

Control Points Need to Be Connected

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

\[
\begin{align*}
\text{r1} & \leftarrow \text{r0} + 10 \\
\text{r4} & \leftarrow \text{r1} + 17 \\
\text{r1} & \leftarrow \ldots
\end{align*}
\]

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

Use strategy from Princeton Pipeline:

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 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
...

0x4 Add

PC

addr inst IR

Inst Memory

nop

IR

IR

31

IR

ALU

rs1 rs2 rd1

ws wd rd2 GPRs

Imm Ext

Y

MD1

MD2

ALU

r data

w data

r data

Memory

Data

w data
Stalled Stages and Pipeline Bubbles

(time)

(I_1) \( r_1 \leftarrow (r_0) + 10 \)
(I_2) \( r_4 \leftarrow (r_1) + 17 \)
(I_3)
(I_4)
(I_5)

(resource usage)

(nop \Rightarrow \text{pipeline bubble})

March 4, 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?  
not every instruction writes a register \( \Rightarrow \) we  
not every instruction reads a register \( \Rightarrow \) re
Thank you!