Instruction Pipelining

Hyun Ryong (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 Sept. 29
• Please view the Pin tutorial video posted on Piazza
• Contact Nikola (in charge of labs) if you cannot get access to lab machines
Announcements

• Lab 1 released later today
  – Designing 3 different cache models using Pin
  – Due Sept. 29
Announcements

• Lab 1 released later today
  – Designing 3 different cache models using Pin
  – Due Sept. 29

• Please view the Pin tutorial video posted on Piazza
Announcements

• Lab 1 released later today
  – Designing 3 different cache models using Pin
  – Due Sept. 29

• Please view the Pin tutorial video posted on Piazza

• Contact Nikola (in charge of labs) if you cannot get access to lab machines
Reminder: Harvard-Style Single-Cycle Datapath for RISC-V
We will assume

- Clock period is sufficiently long for all of the following steps to be “completed”:
  1. instruction fetch
  2. decode and register fetch
  3. ALU operation
  4. data fetch if required
  5. register write-back setup time

\[ t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB} \]

- At the rising edge of the following clock, the PC, the register file and the memory are updated
Datapath for Memory Instructions

Should program and data memory be separate?

*Harvard style: separate* (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

*Princeton style: the same* (von Neumann’s influence)
- single read/write memory for program and data
Should program and data memory be separate?

*Harvard style: separate* (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

- Note:
  There must be a way to load the program memory

*Princeton style: the same* (von Neumann’s influence)
- single read/write memory for program and data
Datapath for Memory Instructions

Should program and data memory be separate?

*Harvard style: separate* (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

- Note:
  There must be a way to load the program memory

*Princeton style: the same* (von Neumann’s influence)
- single read/write memory for program and data

- Note:
  Executing a Load or Store instruction requires accessing the memory more than once
What problem arises if instructions and data reside in the same memory?
Princeton Microarchitecture
Datapath & Control
Princeton Microarchitecture
Datapath & Control
Two-State Controller:
Princeton Architecture

**fetch phase**
- AddrSrc = PC
- IRen = on
- PCen = off
- Wen = off

**execute phase**
- AddrSrc = ALU
- IRen = off
- PCen = on
- Wen = on

A flipflop can be used to remember the phase
Hardwired Controller: 
Princeton Architecture

old combinational logic (Harvard)

IR -> op code & funct

S -> 1-bit Toggle FF

new combinational logic

IR, ImmSel, BSrc, ALUFunct, WBSrc, ...

MemWrite, RegWrite, PCen, IRen, AddrSrc

Wen

1-bit Toggle FF
I-fetch / Execute
Clock Rate vs CPI

$t_{C-Princeton} > \max \{t_M, t_{RF} + t_{ALU} + t_M + t_{WB}\}$
$t_{C-Princeton} > t_{RF} + t_{ALU} + t_M + t_{WB}$
$t_{C-Harvard} > t_M + t_{RF} + t_{ALU} + t_M + t_{WB}$

Suppose $t_M >> t_{RF} + t_{ALU} + t_{WB}$

$t_{C-Princeton} = 0.5 \times t_{C-Harvard}$

$CPI_{Princeton} = 2$
$CPI_{Harvard} = 1$

No difference in performance!

Is it possible to design a controller for the Princeton architecture with CPI < 2?

$CPI = \text{Clock cycles Per Instruction}$
Princeton Microarchitecture (redrawn)

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

Overlapped execution

fetch phase

execute phase
Can we overlap instruction fetch and execute?
Can we overlap instruction fetch and execute?

Yes, unless IR contains a Load or Store
Princeton Microarchitecture
Overlapped execution

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

Which action should be prioritized?
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
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?
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
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?
Stalling the instruction fetch
Princeton Microarchitecture

fetch phase

execute phase

stall?
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)
When stall condition is indicated

- don’t fetch a new instruction and don’t change the PC
When stall condition is indicated
  • *don’t fetch a new instruction and don’t change the PC*
When stall condition is indicated
  • \textit{don’t fetch a new instruction and don’t change the PC}
  • \textit{insert a nop in the IR}
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)*
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

![Diagram showing the fetch and execute phases of a microarchitecture]

- **Fetch phase**
  - PC
  - Add
  - Memory
  - addr
  - wdata

- **Execute phase**
  - IR
  - ALU
  - Imm Ext
  - GPRs
  - rs1
  - rs2
  - rd1
  - we
  - wdata
  - Jump?
  - nop

**Jump?**
Need to stall on branches
Princeton Microarchitecture

When IR contains a jump or taken branch
• no “structural conflict” for the memory

fetch phase

execute phase
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
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
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
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*
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

- ALUFunc
- BSrc
- ALU
- GPRs
- rs1
- rs2
- rd1
- Imm
- regGen
- Zero?
- WBSrc
- MemWrite
- AddrSrc
- pc+4
- Add
- WBSrc
- pc
- IR
- noc
- IRsrc
- Opcode
- ImmSel
- zero?
- stall
- MAddrsr
- we
- Data
- Memory
- wdata
Pipelined Princeton Microarchitecture

![Diagram of Princeton Microarchitecture](image_url)
### Hardwired Control Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Stall</th>
<th>Imm Sel</th>
<th>BSrc</th>
<th>ALU</th>
<th>MemW</th>
<th>WBSrc</th>
<th>RegWr</th>
<th>PC Src1</th>
<th>PC Src2</th>
<th>IRSrc</th>
<th>MAddr Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>no</td>
<td>*</td>
<td>Reg</td>
<td>funct</td>
<td>no</td>
<td>ALU</td>
<td>yes</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>ALUi</td>
<td>no</td>
<td>SXT(imm [11:0])</td>
<td>Imm</td>
<td>funct</td>
<td>no</td>
<td>ALU</td>
<td>yes</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>LW</td>
<td>yes</td>
<td>SXT(imm [11:0])</td>
<td>Imm</td>
<td>+</td>
<td>no</td>
<td>Mem</td>
<td>yes</td>
<td>pc+4</td>
<td>pc</td>
<td>nop</td>
<td>ALU</td>
</tr>
<tr>
<td>SW</td>
<td>yes</td>
<td>SXT(imm [11:0])</td>
<td>Imm</td>
<td>+</td>
<td>yes</td>
<td>*</td>
<td>no</td>
<td>pc+4</td>
<td>pc</td>
<td>nop</td>
<td>ALU</td>
</tr>
<tr>
<td>BRtaken</td>
<td>yes</td>
<td>ImmB</td>
<td>Reg</td>
<td>funct</td>
<td>no</td>
<td>*</td>
<td>no</td>
<td>br</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>BRnotTaken</td>
<td>no</td>
<td>ImmB</td>
<td>Reg</td>
<td>funct</td>
<td>no</td>
<td>*</td>
<td>no</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
<tr>
<td>JALR</td>
<td>yes</td>
<td>ImmI</td>
<td>Imm</td>
<td>+</td>
<td>no</td>
<td>pc+4</td>
<td>yes</td>
<td>jt</td>
<td>npc</td>
<td>nop</td>
<td>*</td>
</tr>
<tr>
<td>JAL</td>
<td>yes</td>
<td>ImmU</td>
<td>Imm</td>
<td>*</td>
<td>no</td>
<td>pc+4</td>
<td>yes</td>
<td>br</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>*</td>
<td>no</td>
<td>pc+4</td>
<td>npc</td>
<td>mem</td>
<td>pc</td>
</tr>
</tbody>
</table>

BSrc = Reg / Imm; IRSrc = nop/mem; MAddrSrc = pc/ALU; WBSrc = ALU / Mem / pc+4
PCSrc1 = pc+4 / br / rind; PCSrc2 = npc/pc;
Pipelined Princeton Architecture

**Clock:** \( t_{\text{C-Princeton}} > t_{\text{RF}} + t_{\text{ALU}} + t_{M} + t_{\text{WB}} \)

**CPI:** \( (1 - f) + 2f \) cycles per instruction where \( f \) is the fraction of instructions that cause a stall
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.

\[ \text{Clock period} > \max \{ t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW} \} \]

However, CPI will increase unless instructions are pipelined.
Pipelined Datapath

Clock period can be reduced by dividing the execution of an instruction into multiple cycles:

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

However, CPI will increase unless instructions are pipelined.

Diagram of the pipelined datapath.
Clock period can be reduced by dividing the execution of an instruction into multiple cycles.

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

However, CPI will increase unless instructions are pipelined.
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

\[ t_c > \max \{ t_{\text{IM}}, t_{\text{RF}}, t_{\text{ALU}}, t_{\text{DM}}, t_{\text{RW}} \} = t_{\text{DM}} \]
Alternative Pipelining

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

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

Write-back stage takes much less time than other stages.
Alternative Pipelining

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

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

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

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

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

---
# Maximum Speedup by Pipelining

**Assumptions**

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

<table>
<thead>
<tr>
<th>Unpipelined $t_C$</th>
<th>Pipelined $t_C$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

September 15, 2023  
MIT 6.5900 (6.823) Fall 2023  
R02-22
# Maximum Speedup by Pipelining

## Assumptions

<table>
<thead>
<tr>
<th>Assumption</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></td>
<td></td>
<td>27</td>
</tr>
</tbody>
</table>

4-stage pipeline
# Maximum Speedup by Pipelining

## Assumptions

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

<table>
<thead>
<tr>
<th>4-stage pipeline</th>
<th>Unpipelined ( t_C )</th>
<th>Pipelined ( t_C )</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>27</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

---

September 15, 2023

MIT 6.5900 (6.823) Fall 2023
# Maximum Speedup by Pipelining

**Assumptions**

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_C )</td>
<td>( t_C )</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.7</td>
</tr>
</tbody>
</table>
Assumptions

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

\[ t_C = 27 \]

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

\[ t_C = 10 \]

Speedup

\[ \text{Speedup} = \frac{t_C^{\text{unpipelined}}}{t_C^{\text{pipelined}}} = \frac{27}{10} = 2.7 \]
# 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></td>
<td></td>
</tr>
</tbody>
</table>
# Maximum Speedup by Pipelining

## Assumptions

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_C )</td>
<td>( t_C )</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>10</td>
<td>2.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_C )</td>
<td>( t_C )</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
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>4-stage pipeline</td>
<td></td>
<td></td>
<td></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>4-stage pipeline</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 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></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 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></td>
<td></td>
</tr>
</tbody>
</table>
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></td>
</tr>
</tbody>
</table>
## Maximum Speedup by Pipelining

### Assumptions

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

\[ \frac{27}{10} = 2.7 \]

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

\[ \frac{25}{10} = 2.5 \]

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

\[ \frac{25}{5} = 5.0 \]
# Maximum Speedup by Pipelining

## Assumptions

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>10</td>
<td>2.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>2.5</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Unpipelined</th>
<th>Pipelined</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>5</td>
<td>5.0</td>
</tr>
</tbody>
</table>

*What seems to be the message here?*
## Maximum Speedup by Pipelining

### Assumptions

1. $t_{IM} = t_{DM} = 10$, 
   $t_{ALU} = 5$, 
   $t_{RF} = t_{RW} = 1$
   4-stage pipeline
   
   Unpipelined $t_C$: 27
   Pipelined $t_C$: 10
   Speedup: 2.7

2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$
   4-stage pipeline
   
   Unpipelined $t_C$: 25
   Pipelined $t_C$: 10
   Speedup: 2.5

3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$
   5-stage pipeline
   
   Unpipelined $t_C$: 25
   Pipelined $t_C$: 5
   Speedup: 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)**: Fetches the next instruction from memory.
- **Decode, Reg. Fetch (ID)**: Decodes the instruction and fetches registers.
- **Execute (EX)**: Performs the execute phase, including ALU operations.
- **Memory (MA)**: Reads or writes data from memory.
- **Write Back (WB)**: Writes back the results to memory.

Diagram includes components such as PC, IR, adder, ALU, and memory access paths.
5-Stage Pipelined Execution

Instruction Flow Diagram

I-Fetch (IF)

Decode, Reg. Fetch (ID)

Execute (EX)

Memory (MA)

Write-Back (WB)

Time

t0  t1  t2  t3  t4  t5  t6  t7  . . .
5-Stage Pipelined Execution

Instruction Flow Diagram

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

**Instruction Flow**:
- **time**
- **instruction1**
- **t0**
- **t1**
- **t2**
- **t3**
- **t4**
- **t5**
- **t6**
- **t7**

**Units**: IF, ID, EX, MA, WB

**Symbols**: PC, IR, Inst. Memory, Addr, Rdata, ALU, Imm Gen, GPRs, Addr, Rdata, Memory, Write-Back, Add
5-Stage Pipelined Execution

**Instruction Flow Diagram**

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
<td>WB</td>
<td>WB</td>
<td>WB</td>
</tr>
<tr>
<td>t0</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>t4</td>
<td>t5</td>
<td>t6</td>
</tr>
</tbody>
</table>

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

**Time Sequence**
- Instruction 1
  - t0: Fetch
  - t1: Decode
  - t2: Execute
  - t3: Memory
  - t4: Write
- Instruction 2
  - t5: Fetch
  - t6: Decode
  - t7: Execute
  - t8: Memory
  - t9: Write

**Memory Access**
- Memory Address (addr)
- Memory Data (wdata)

**ALU Operations**
- ALU Imm Gen
- ALU Gen
- ALU Add

**Register Access**
- GPRs
- GPRs
- GPRs

**Write-Back**
- Write Back (WB)
- Write Back (WB)
- Write Back (WB)

**Instruction Flow**
- Instruction Fetch (IF)
- Instruction Decode (ID)
- Instruction Execute (EX)
- Instruction Memory (MA)
- Instruction Write-Back (WB)
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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;1&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>IF&lt;sub&gt;2&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;2&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>IF&lt;sub&gt;3&lt;/sub&gt;</td>
<td>ID&lt;sub&gt;3&lt;/sub&gt;</td>
<td>EX&lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PC → Addr → Inst. Memory

Addr → IF → IR

IR → ID → EX → MA → WB

Address Memory

Address GPRs

Imm Gen

ALU

ALU Imm Gen

Addition

Write-Back

Load, Store

R02 - 23
5-Stage Pipelined Execution

Instruction Flow Diagram

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

**Time Table**

<table>
<thead>
<tr>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Instruction 3</th>
<th>Instruction 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0 (IF₁)</td>
<td>t1 (ID₁)</td>
<td>t2 (EX₁)</td>
<td>t3 (MA₁)</td>
</tr>
<tr>
<td>t1 (IF₂)</td>
<td>t2 (ID₂)</td>
<td>t3 (MA₂)</td>
<td>t4 (WB₁)</td>
</tr>
<tr>
<td>t2 (IF₃)</td>
<td>t3 (ID₃)</td>
<td>t4 (MA₂)</td>
<td>t5 (WB₂)</td>
</tr>
<tr>
<td>t3 (IF₄)</td>
<td>t4 (ID₄)</td>
<td>t5 (MA₃)</td>
<td>t6 (WB₃)</td>
</tr>
<tr>
<td>t4</td>
<td>t5</td>
<td>t6</td>
<td>t7</td>
</tr>
</tbody>
</table>
5-Stage Pipelined Execution

Instruction Flow Diagram

- **I-Fetch (IF)**
  - **instruction1**
  - **instruction2**
  - **instruction3**
  - **instruction4**
  - **instruction5**

- **Decode, Reg. Fetch (ID)**
  - **t0**
  - **t1**

- **Execute (EX)**
  - **t2**
  - **t3**
  - **t4**
  - **t5**

- **Memory (MA)**
  - **t6**
  - **t7**

- **Write-Back (WB)**
  - **t8**

- **time**
  - **t0**
  - **t1**
  - **t2**
  - **t3**
  - **t4**
  - **t5**
  - **t6**
  - **t7**

- **Add**
  - **0x4**

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

- **PC**

- **IR**

- **ALU**
  - **addr**
  - **rdata**
  - **wdata**

- **GPRs**
  - **rs1**
  - **rs2**
  - **rd1**
  - **ws**
  - **wd**
  - **rd2**

- **Imm Gen**

- **Memory**
  - **rdata**
  - **wdata**

- **Write-Back (WB)**
5-Stage Pipelined Execution

Instruction Flow Diagram

- **I-Fetch (IF)**: Fetches the next instruction from memory.
- **Decode, Reg. Fetch (ID)**: Decodes the instruction and fetches registers.
- **Execute (EX)**: Performs the operation specified by the instruction.
- **Memory (MA)**: Accesses memory for data or writes data to memory.
- **Write-Back (WB)**: Writes back computed results to memory.

**Instruction Flow**

- **time**: t0, t1, t2, t3, t4, t5, t6, t7, ...
- **Instruction**: instruction1 through instruction5

**Pipeline Stages**

- IF: Fetch
- ID: Decode, Register Fetch
- EX: Execute
- MA: Memory
- WB: Write-Back

**Data Path**

- **PC**: Program Counter
- **Addr**: Address
- **Inst. Memory**: Instruction Memory
- **IR**: Instruction Register
- **Imm Gen**: Immediate Generator
- **ALU**: Arithmetic Logic Unit
- **GPRs**: General Purpose Registers
- **Memory**: Data Memory
- **Write-Back**: Write-Back Buffer

**Example Instruction**

- **0x4 Add**
- **addr**: Address
- **data**: Data
- **PC**: Program Counter
- **result**: Result of addition

**Pipeline Phases**

- **IF**: Fetch
- **ID**: Decode, Register Fetch
- **EX**: Execute
- **MA**: Memory Access
- **WB**: Write-Back

**Pipelined Execution Timeline**

- **t0**: IF1
- **t1**: ID1
- **t2**: EX1
- **t3**: MA1
- **t4**: WB1
- **t5**: IF2
- **t6**: ID2
- **t7**: EX2
- **t8**: MA2
- **t9**: WB2
- **t10**: IF3
- **t11**: ID3
- **t12**: EX3
- **t13**: MA3
- **t14**: WB3
- **t15**: IF4
- **t16**: ID4
- **t17**: EX4
- **t18**: MA4
- **t19**: WB4
- **t20**: IF5
- **t21**: ID5
- **t22**: EX5
- **t23**: MA5
- **t24**: WB5

**Notes**

- Each cycle represents one instruction.
- Pipelining allows multiple instructions to be processed simultaneously.
- Data dependencies and cache misses can affect pipeline performance.

**References**

- MIT 6.823 Spring 2023
- September 15, 2023
- MIT 6.5900 (6.823) Fall 2023
- R02-23
5-Stage Pipelined Execution

Resource Usage Diagram

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

Resources

0x4 0x4
Addr  Addr
Inst. Memory  IR

PC  addr  rdata

R02 24

September 15, 2023  MIT 6.5900 (6.823) Fall 2023
5-Stage Pipelined Execution

Resource Usage Diagram

- **I-Fetch (IF):**
  - **PC**
  - **Addr**
  - **Rdata**
  - **Inst. Memory**

- **Decode, Reg. Fetch (ID):**
  - **IR**
  - **Imm Gen**
  - **WE**
  - **WE**

- **Execute (EX):**
  - **ALU**
  - **Addr**
  - **Rdata**
  - **Mem Memory**

- **Write-Back (WB):**
  - **Write-Back**

---

September 15, 2023

MIT 6.5900 (6.823) Fall 2023

R02-24
5-Stage Pipelined Execution

Resource Usage Diagram

![Diagram of 5-stage pipelined execution with stages: I-Fetch (IF), Decode, Reg. Fetch (ID), Execute (EX), Memory (MA), Write Back (WB). Each stage has time points (t0 to t7) and resource usage diagram with symbols for instructions (I1 to I5), data (addr, rdata), and control signals (we, rs1, rs2, rd1, ws, wd, rd2).]
5-Stage Pipelined Execution

Resource Usage Diagram

I-Fetch (IF)

IF

ID

Decode, Reg. Fetch (ID)

Execute (EX)

Memory (MA)

Write-Back (WB)

Resources

<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>
<th>. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td>I₈</td>
<td>. . .</td>
</tr>
<tr>
<td>ID</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td></td>
<td></td>
<td></td>
<td>. . .</td>
</tr>
</tbody>
</table>
5-Stage Pipelined Execution

Resource Usage Diagram

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

- **Decode, Reg. Fetch (ID)**
  - Resources: ID
  - Time: I1, I2, I3, I4, I5

- **Execute (EX)**
  - Resources: EX
  - Time: I1, I2, I3, I4, I5

- **Memory (MA)**
  - Resources: Memory
  - Time: ...

- **Write-Back (WB)**
  - Resources: Write-Back

Diagram includes symbols for PC, Addr, Inst. Memory, IR, ALU, Imm Gen, Add, and other components relevant to the pipeline stages.
5-Stage Pipelined Execution

Resource Usage Diagram

- **IF (I-Fetch)**: Address Generation, Instruction Fetch
- **ID (Decode, Reg. Fetch)**: Instruction Decode, Register Fetch
- **EX (Execute)**: ALU Operation, Result Generation
- **MA (Memory)**: Memory Access, Data Fetch
- **WB (Write-Back)**: Store Result, Update PC

### Resources
- **IF**:
  - **time**: t0, t1, t2, t3, t4, t5, t6, t7, ...
  - **resources**: I₁, I₂, I₃, I₄, I₅

- **ID**: I₁, I₂, I₃, I₄, I₅

- **EX**: I₁, I₂, I₃, I₄, I₅

- **MA**: I₁, I₂, I₃, I₄, I₅
5-Stage Pipelined Execution

Resource Usage Diagram

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

Resources:
- IF
- ID
- EX
- MA
- WB

Time:
- t0
- t1
- t2
- t3
- t4
- t5
- t6
- t7

Resources:
- IR
- PC
- Addr
- Rdata
- Inst.
- Memory
- ALU
- Imm Gen
- GPRs

Add
- 0x4

September 15, 2023
5-Stage Pipelined Execution
Resource Usage Diagram

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

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>t0</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>t4</td>
</tr>
<tr>
<td></td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I₁</td>
<td>I₂</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I₁</td>
</tr>
</tbody>
</table>

September 15, 2023
Pipelined Execution

ALU Instructions

Not quite correct!

We need an Instruction Reg (IR) for each stage
Not quite correct!
Not quite correct!

We need an Instruction Reg (IR) for each stage
Not quite correct!

We need an Instruction Reg (IR) for each stage
Pipelined Execution

ALU Instructions

Not quite correct!

We need an Instruction Reg (IR) for each stage
Not quite correct!

We need an Instruction Reg (IR) for each stage
Pipelined RISC-V Datapath

without jumps

What else is needed?
Pipelined RISC-V Datapath

without jumps

What else is needed?

Control Points Need to Be Connected
Pipelined RISC-V Datapath
without jumps

What else is needed?

Control Points Need to Be Connected

September 15, 2023
MIT 6.5900 (6.823) Fall 2023
Pipelined RISC-V Datapath

without jumps

What else is needed?

Control Points Need to Be Connected
Pipelined RISC-V Datapath
without jumps

What else is needed?

Control Points Need to Be Connected
Pipelined RISC-V 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: data hazard
  - Dependence may be for a data calculation: data hazard
  - Dependence may be for calculating the next PC: control hazard (branches, interrupts)
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
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
...
Data Hazards

r1 ← r0 + 10
r4 ← r1 + 17

...
Data Hazards

...  
r1 ← r0 + 10  
r4 ← r1 + 17  
...
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
Later stages provide dependence information to earlier stages which can *stall instructions*
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can *stall instructions*
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can stall instructions
Later stages provide dependence information to earlier stages which can stall instructions.
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can *stall instructions*
Feedback to Resolve Hazards

Later stages provide dependence information to earlier stages which can *stall instructions*.
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can *stall instructions*
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can *stall instructions*
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

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

...
Resolving Data Hazards by Stalling

... r1 ← r0 + 10
r4 ← r1 + 17
...
Resolving Data Hazards by Stalling

... r1 ← r0 + 10
r4 ← r1 + 17
...

September 15, 2023
Resolving Data Hazards by Stalling

... r1 ← r0 + 10
r4 ← r1 + 17
...

Stall Condition

...
Stalled Stages and Pipeline Bubbles

\[\begin{align*}
&\text{time} \quad t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad \ldots \\
&\text{IF} \quad I_1 \quad I_2 \quad I_3 \quad I_3 \quad I_3 \quad I_3 \quad I_4 \quad I_5 \\
&\text{ID} \quad I_1 \quad I_2 \quad I_2 \quad I_2 \quad I_2 \quad I_2 \quad I_4 \quad I_5 \\
&\text{EX} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad I_2 \quad I_3 \quad I_4 \quad I_5 \\
&\text{MA} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad I_2 \quad I_3 \quad I_4 \quad I_5 \\
&\text{WB} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad \text{nop} \quad I_4 \quad I_5 \\
\end{align*}\]
Stalled Stages and Pipeline Bubbles

\[
time
\begin{array}{ccccccccc}
t0 & t1 & t2 & t3 & t4 & t5 & t6 & t7 & \ldots
\end{array}
\]
Stalled Stages and Pipeline Bubbles

\(\text{time}\)
\begin{align*}
t0 & \quad t1 & \quad t2 & \quad t3 & \quad t4 & \quad t5 & \quad t6 & \quad t7 & \ldots. \\
(I_1) & \quad r1 & \leftarrow (r0) + 10 & \quad IF_1 & \quad ID_1 & \quad EX_1 & \quad MA_1 & \quad WB_1
\end{align*}
Stalled Stages and Pipeline Bubbles

\[ \begin{array}{cccccccc}
\text{time} & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\
(I_1) & r_1 & \leftarrow & (r_0) + 10 & \text{IF}_1 & \text{ID}_1 & \text{EX}_1 & \text{MA}_1 & \text{WB}_1 \\
(I_2) & r_4 & \leftarrow & (r_1) + 17 & \text{IF}_2 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_2 & \text{MA}_2 & \text{WB}_2
\end{array} \]
Stalled Stages and Pipeline Bubbles

\[ \text{time} \]
\[
\begin{array}{cccccccc}
t0 & t1 & t2 & t3 & t4 & t5 & t6 & t7 & \ldots \ldots \\
(\text{I}_1) & r1 \leftarrow (r0) + 10 & \text{IF}_1 & \text{ID}_1 & \text{EX}_1 & \text{MA}_1 & \text{WB}_1 \\
(\text{I}_2) & r4 \leftarrow (r1) + 17 & \text{IF}_2 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_2 & \text{MA}_2 & \text{WB}_2 \\
(\text{I}_3) & & \text{IF}_3 & \text{IF}_3 & \text{IF}_3 & \text{IF}_3 & \text{ID}_3 & \text{EX}_3 & \text{MA}_3 & \text{WB}_3
\end{array}
\]
Stalled Stages and Pipeline Bubbles

\[
\begin{array}{cccccccc}
\text{time} & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\
(\text{I}_1) & r_1 & \leftarrow & (r_0) + 10 & \text{IF}_1 & \text{ID}_1 & \text{EX}_1 & \text{MA}_1 & \text{WB}_1 \\
(\text{I}_2) & r_4 & \leftarrow & (r_1) + 17 & \text{IF}_2 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_2 & \text{MA}_2 & \text{WB}_2 \\
(\text{I}_3) & & & & \text{IF}_3 & \text{ID}_3 & \text{ID}_3 & \text{ID}_3 & \text{EX}_3 & \text{MA}_3 & \text{WB}_3 \\
(\text{I}_4) & & & & & & & & \text{IF}_4 & \text{ID}_4 & \text{EX}_4 & \text{MA}_4 & \text{WB}_4
\end{array}
\]
Stalled Stages and Pipeline Bubbles

\[
\begin{array}{ccccccccc}
\text{time} & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & \ldots \\
(I_1) & r_1 & \leftarrow & (r_0) & + & 10 & \text{IF}_1 & & & \\
(I_2) & r_4 & \leftarrow & (r_1) & + & 17 & \text{IF}_2 & \text{ID}_1 & \text{EX}_1 & \text{MA}_1 & \text{WB}_1 \\
(I_3) & & & \text{IF}_2 & \text{ID}_2 & & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_2 & \text{MA}_2 & \text{WB}_2 \\
(I_4) & & & & \text{IF}_3 & \text{ID}_3 & & \text{ID}_3 & \text{ID}_3 & \text{ID}_3 & \text{EX}_3 & \text{MA}_3 & \text{WB}_3 \\
(I_5) & & & & & \text{IF}_4 & \text{ID}_4 & \text{EX}_4 & \text{MA}_4 & \text{WB}_4 \\
\end{array}
\]
Stalled Stages and Pipeline Bubbles

(time)

\( t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad \ldots \)

\( (I_1) \ r_1 \leftarrow (r_0) + 10 \)
\( (I_2) \ r_4 \leftarrow (r_1) + 17 \)
\( (I_3) \)
\( (I_4) \)
\( (I_5) \)
Stalled Stages and Pipeline Bubbles

\[
\begin{align*}
&\text{time} \\
&t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad \ldots \\
&\text{IF}_1 \quad \text{ID}_1 \quad \text{EX}_1 \quad \text{MA}_1 \quad \text{WB}_1 \\
&\text{IF}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \\
&\text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \\
&\text{IF}_4 \quad \text{ID}_4 \quad \text{EX}_4 \quad \text{MA}_4 \quad \text{WB}_4 \\
&\text{IF}_5 \quad \text{ID}_5 \quad \text{EX}_5 \quad \text{MA}_5 \quad \text{WB}_5 \\
\end{align*}
\]

\(r1 \leftarrow (r0) + 10\)  
\(r4 \leftarrow (r1) + 17\)
Stalled Stages and Pipeline Bubbles

Resource Usage

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

**time**

- \(t_0\)
- \(t_1\)
- \(t_2\)
- \(t_3\)
- \(t_4\)
- \(t_5\)
- \(t_6\)
- \(t_7\) 

**stalled stages**

- IF
- ID
- EX
- MA
- WB
Stalled Stages and Pipeline Bubbles

\[
\begin{align*}
\text{time} & \quad t0 \quad t1 \quad t2 \quad t3 \quad t4 \quad t5 \quad t6 \quad t7 \quad \ldots \\
(I_1) \quad r1 & \leftarrow (r0) + 10 \\
(I_2) \quad r4 & \leftarrow (r1) + 17 \\
(I_3) \\
(I_4) \\
(I_5) \\
\end{align*}
\]

\textit{stalled stages}

\[
\begin{align*}
\text{time} & \quad t0 \quad t1 \quad t2 \quad t3 \quad t4 \quad t5 \quad t6 \quad t7 \quad \ldots \\
\end{align*}
\]

Resource Usage
Stalled Stages and Pipeline Bubbles

(time)

\[ t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad \ldots \]

\[ (I_1) \ r_1 \leftarrow (r_0) + 10 \quad \text{IF}_1 \quad \text{ID}_1 \quad \text{EX}_1 \quad \text{MA}_1 \quad \text{WB}_1 \]

\[ (I_2) \ r_4 \leftarrow (r_1) + 17 \quad \text{IF}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{ID}_2 \quad \text{EX}_2 \quad \text{MA}_2 \quad \text{WB}_2 \]

\[ (I_3) \quad \text{stalled stages} \]

\[ (I_4) \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{IF}_3 \quad \text{EX}_3 \quad \text{MA}_3 \quad \text{WB}_3 \]

\[ (I_5) \quad \text{IF}_4 \quad \text{ID}_4 \quad \text{EX}_4 \quad \text{MA}_4 \quad \text{WB}_4 \]

\[ (I_6) \quad \text{IF}_5 \quad \text{ID}_5 \quad \text{EX}_5 \quad \text{MA}_5 \quad \text{WB}_5 \]

Resource Usage

September 15, 2023
Stalled Stages and Pipeline Bubbles

Stalled stages

Resource Usage

September 15, 2023
Stalled Stages and Pipeline Bubbles

```

<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>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1</td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td></td>
</tr>
<tr>
<td>ID1</td>
<td>I1</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td></td>
</tr>
<tr>
<td>EX1</td>
<td>I1</td>
<td>nop</td>
<td>nop</td>
<td>nop</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td></td>
</tr>
<tr>
<td>MA1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stalled stages

(I_1) r1 ← (r0) + 10
(I_2) r4 ← (r1) + 17
(I_3)
(I_4)
(I_5)

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>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>I1</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>I1</td>
<td>nop</td>
<td>nop</td>
<td>nop</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IF, ID, EX

September 15, 2023
Stalled Stages and Pipeline Bubbles

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

Resource Usage

<table>
<thead>
<tr>
<th>Time</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>I_1</td>
<td>I_1</td>
<td>I_1</td>
<td>I_1</td>
</tr>
<tr>
<td>t1</td>
<td>I_2</td>
<td>I_2</td>
<td>I_2</td>
<td>I_2</td>
</tr>
<tr>
<td>t2</td>
<td>I_3</td>
<td>I_3</td>
<td>I_3</td>
<td>I_3</td>
</tr>
<tr>
<td>t3</td>
<td>I_3</td>
<td>I_3</td>
<td>I_3</td>
<td>I_3</td>
</tr>
<tr>
<td>t4</td>
<td>I_4</td>
<td>I_4</td>
<td>I_4</td>
<td>I_4</td>
</tr>
<tr>
<td>t5</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
</tr>
<tr>
<td>t6</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
</tr>
<tr>
<td>t7</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
<td>I_5</td>
</tr>
</tbody>
</table>

\[ \text{stalled stages} \]
Stalled Stages and Pipeline Bubbles

\[ \begin{align*}
(I_1) & \quad r_1 \leftarrow (r_0) + 10 \\
(I_2) & \quad r_4 \leftarrow (r_1) + 17 \\
(I_3) & \\
(I_4) & \\
(I_5) &
\end{align*} \]

Resource Usage

\[ \begin{align*}
\text{time} & \\
t0 & \quad \text{IF}_1 \\
t1 & \quad \text{ID}_1 \\
t2 & \quad \text{EX}_1 \quad \text{MA}_1 \quad \text{WB}_1 \\
t3 & \\
t4 & \\
t5 & \\
t6 & \\
t7 & \ldots \ldots
\end{align*} \]

\[ \begin{align*}
\text{IF} & \quad I_1 \\
\text{ID} & \quad I_1 \quad I_2 \\
\text{EX} & \quad I_1 \quad \text{nop} \quad \text{nop} \\
\text{MA} & \quad I_1 \quad \text{nop} \quad \text{nop} \\
\text{WB} & \quad I_1 \quad \text{nop} \quad \text{nop} \\
\text{IF}_2 & \quad I_2 \\
\text{ID}_2 & \quad I_2 \quad I_2 \\
\text{EX}_2 & \quad I_2 \quad \text{nop} \\
\text{MA}_2 & \quad I_2 \quad \text{nop} \\
\text{WB}_2 & \quad I_2 \quad \text{nop} \\
\text{IF}_3 & \quad I_3 \\
\text{ID}_3 & \quad I_3 \quad I_3 \\
\text{EX}_3 & \quad I_3 \quad \text{nop} \\
\text{MA}_3 & \quad I_3 \quad \text{nop} \\
\text{WB}_3 & \quad I_3 \quad \text{nop} \\
\text{IF}_4 & \quad I_4 \\
\text{ID}_4 & \quad I_4 \quad I_4 \\
\text{EX}_4 & \quad I_4 \quad \text{nop} \\
\text{MA}_4 & \quad I_4 \quad \text{nop} \\
\text{WB}_4 & \quad I_4 \quad \text{nop} \\
\text{IF}_5 & \quad I_5 \\
\text{ID}_5 & \quad I_5 \quad I_5 \\
\text{EX}_5 & \quad I_5 \quad \text{nop} \\
\text{MA}_5 & \quad I_5 \quad \text{nop} \\
\text{WB}_5 & \quad I_5 \quad \text{nop}
\end{align*} \]
Stalled Stages and Pipeline Bubbles

\[ (I_1) \ r_1 \leftarrow (r_0) + 10 \]
\[ (I_2) \ r_4 \leftarrow (r_1) + 17 \]
\[ (I_3) \]
\[ (I_4) \]
\[ (I_5) \]
Stalled Stages and Pipeline Bubbles

(time
t0  t1  t2  t3  t4  t5  t6  t7  . . . .
(I_1) r1 \leftarrow (r0) + 10
(I_2) r4 \leftarrow (r1) + 17
(I_3)
(I_4)
(I_5)

Resource Usage

(time
t0  t1  t2  t3  t4  t5  t6  t7  . . . .
IF  I_1  I_2  I_3  I_3  I_3  I_3  I_4  I_5
ID  I_1  I_2  I_2  I_2  I_2  I_2  I_3  I_4  I_5
EX  I_1  nop  nop  nop  nop  nop  I_2  I_3  I_4  I_5
MA  I_1  nop  nop  nop  nop  nop  I_2  I_3  I_4  I_5
WB  I_1  nop  nop  nop  nop  nop  I_2  I_3  I_4  I_5

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

Should we always stall if the rs field(s) matches some rd?
Should we always stall if the rs field(s) matches some rd?

not every instruction writes a register ⇒ we
Stall Control Logic
ignoring jumps & branches

Should we always stall if the rs field(s) matches some rd?
not every instruction writes a register ⇒ we
not every instruction reads a register ⇒ re
Should we always stall if the rs field(s) matches some rd?

not every instruction writes a register ⇒ we
not every instruction reads a register ⇒ re

September 15, 2023
Should we always stall if the rs field(s) matches some rd?

not every instruction writes a register ⇒ we
not every instruction reads a register ⇒ re
Stall Control Logic
ignoring jumps & branches

Should we always stall if the rs field(s) matches some rd?
not every instruction writes a register \(\Rightarrow\) we
not every instruction reads a register \(\Rightarrow\) re
Thank you!