Laundry Pipelined

Task A
- 6pm: Wash
- 7pm: Dry
- 8pm: Fold

Task B
- 6pm: Wash
- 7pm: Dry
- 8pm: Fold

Task C
- 6pm: Wash
- 7pm: Dry
- 8pm: Fold

Task D
- 6pm: Wash
- 7pm: Dry
- 8pm: Fold
• Theoretical speed-up: 3 times

• Actual speed-up in example: 2 times
  – sequential: 1:30+1:30+1:30+1:30 = 6 hours
  – pipelined: 1:30+0:30+0:30+0:30 = 3 hours

• Many tasks → speed-up approaches theoretical limit
mips instruction pipeline
MIPS Pipeline

• Fetch instruction from memory

• Read registers and decode instruction
  (note: registers are always encoded in same place in instruction)

• Execute operation OR calculate an address

• Access an operand in memory

• Write result into a register
## Time for Instructions

- Breakdown for each type of instruction

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>Instr. fetch</th>
<th>Register read</th>
<th>ALU oper.</th>
<th>Data access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load word (lw)</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100ps</td>
<td>800ps</td>
</tr>
<tr>
<td>Store word (lw)</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format (add)</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td></td>
<td>100ps</td>
<td>600ps</td>
</tr>
<tr>
<td>Branch (beq)</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>
Pipeline Execution

```
lw $t1, 100($t0)
lw $t2, 104($t0)
lw $t3, 108($t0)
```

Instruction Fetch  Reg. read  ALU  Data access  Reg. write

Instruction Fetch  Reg. read  ALU  Data access  Reg. write

Instruction Fetch  Reg. read  ALU  Data access  Reg. write

Instruction Fetch  Reg. read  ALU  Data access  Reg. write
• Theoretical speed-up: 4 times

• Actual speed-up in example: 1.71 times
  - sequential: $800\text{ps} + 800\text{ps} + 800\text{ps} = 2400\text{ps}$
  - pipelined: $1000\text{ps} + 200\text{ps} + 200\text{ps} = 1400\text{ps}$

• Many tasks $\rightarrow$ speed-up approaches theoretical limit
Design for Pipelining

• All instructions are 4 bytes
  → easy to fetch next instruction

• Few instruction formats
  → parallel op decode and register read

• Memory access limited to load and store instructions
  → stage 3 used for memory access, otherwise operation execution

• Words aligned in memory
  → able to read in one instruction
  (aligned = memory address multiple of 4)
hazards
Hazards

• Hazard = next instruction cannot be executed in next clock cycle

• Types
  – structural hazard
  – data hazard
  – control hazard
**Structural Hazard**

- **Definition:** instructions overlap in resource use in same stage

- **For instance:** memory access conflict

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>FETCH</td>
<td>DECODE</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td>ALU</td>
<td>REGISTER</td>
</tr>
<tr>
<td>i2</td>
<td>FETCH</td>
<td>DECODE</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td>ALU</td>
<td>REGISTER</td>
</tr>
</tbody>
</table>
```

- **MIPS designed to avoid structural hazards**
Data Hazard

- Definition: instruction waits on result from prior instruction

- Example
  
  add $s0, $t0, $t1
  sub $t0, $s0, $t3

  - add instruction writes result to register $s0 in stage 5
  - sub instruction reads $s0 in stage 2

  \( \implies \) Stage 2 of sub has to be delayed

- We overcome this in hardware
Graphical Representation

- **IF**: instruction fetch
- **ID**: instruction decode
- **EX**: execution
- **MEM**: memory access
- **WB**: write-back

```
add $s0,$t0,$t1
```

```
200 400 600 800 1000
```
Add and Subtract

- Add wiring to circuit to directly connect output of ALU for next instruction
Load and Subtract

- Add wiring from memory lookup to ALU
- Still 1 cycle unused: "pipeline stall" or "bubble"
Reorder Code

• Code with data hazard

\[
\begin{align*}
\text{lw } & \quad \texttt{$t1, 0($t0)} \\
\text{lw } & \quad \texttt{$t2, 4($t0)} \\
\text{add } & \quad \texttt{$t3, $t1, $t2} \\
\text{sw } & \quad \texttt{$t3, 12($t0)} \\
\text{lw } & \quad \texttt{$t4, 8($t0)} \\
\text{add } & \quad \texttt{$t5, $t1, $t4} \\
\text{sw } & \quad \texttt{$t5, 16($t0)}
\end{align*}
\]

• Reorder code (may be done by compiler)

• Load instruction now completed in time
Control Hazard

• Also called branch hazard

• Selection of next instruction depends on outcome of previous

• Example

  add $s0, $t0, $t1
  beq $s0, $s1, ff40
  sub $t0, $s0, $t3

  – sub instruction only executed if branch condition fails
  → cannot start until branch condition result known
Branch Prediction

• Assume that branches are never taken
  → full speed if correct

• More sophisticated
  – keep record of branch taken or not
  – make prediction based on history
pipelined data path
Datapath

IF: Instruction Fetch
ID: Instruction decoder
EX: Execute / address calculate
MEM: Memory access
WB: Write Back

Selector
PC
Add

Instruction
Memory

Address

Selector

Read register 1
Read data 1
Read data 2
Write register
Write data

1

4

Add

Sign extended

Shift Left

Zero

Selector

ALU
Result

Address
Read data

Selector

Data
Memory
Write data
Pipelined Datapath

IF: Instruction Fetch

ID: Instruction decoder
- register file read

EX: Execute / address calculate
- ALU
- Add
- Instruction
- Memory
- Address
- Instruction
- Registers
- Read register 1
- Read register 2
- Write register
- Write data

MEM: Memory access
- Data
- Memory
- Address
- Read data
- Write data

WB: Write Back
- Selector
- PC
- Address Memory
- Instruction
- Instruction decoder
- Register file read
- MEM: 
- Write Back
- Selector
load
Load: Stage 2

IF: Instruction Fetch

ID: Instruction decoder
    register file read

EX: Execute /
    address calculate

MEM: Memory access

WB: Write Back

Registers

Instruction

SelectPC

Address

Instruction Memory

Instruction

Write register

Read register 1

Read register 2

Read data 1

Read data 2

Write data

4

Add

Sign extended

Shift Left

ADD

Zero

ALU Result

Address

Data Memory

Read data

Write data

Selector

Selector
Load: Stage 3

IF: Instruction Fetch
- Selector
- PC
- Address
- Instruction Memory
- Instruction

ID: Instruction decoder
- Register file read
- Add
- 4
- Address
- Instruction Memory
- Instruction
- Registers
- Read register 1
- Read data 1
- Read register 2
- Write register
- Read data 2
- Write data
- Sign extended

EX: Execute / address calculate
- Shift Left
- ADD
- Zero
- ALU Result
- Selector
- Address
- Read data
- Data Memory
- Write data

MEM: Memory access
- Write data

WB: Write Back
- Selector
Load: Stage 4

IF: Instruction Fetch
ID: Instruction decoder
MEM: Memory access
WB: Write Back

Selector
Add
Address
Instruction Memory
Instruction

PC

4

Sign extended
Shift Left
Zero

ADD

Register
Read register 1
Read data 1

Read register 2
Read data 2

Write register
Write data

Data Memory

Address
Read data

Write data

Selector

 selector

 selector

 selector
Load: Stage 5

IF: Instruction Fetch
ID: Instruction decoder
MEM: Memory access
WB: Write Back

Add

Selector
PC

Address Memory
Instruction

Instruction Fetch
ID: Instruction decoder
register file read
MEM: Memory access

EX: Execute /
address calculate

Zero

ALU
Add
Instruction
Memory
Address
Instruction
Registers
Read 
register 1
Read 
register 2
Write 
register
Write 
data

Shift Left

Selector

Read data 1
Read data 2

Selector

Read data
Data Memory
Write data

Add

Selector
store
Store: Stage 1
Store: Stage 2

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Selector PC

Add

Sign extended

Shift Left

Zero

ALU Result

Address

Read

Selector

Instruction Memory

Instruction

PC 4

Add

Read register 1
Read register 2
Write register
Write data

Read data 1
Read data 2

Write data

Data Memory

Read data

Write data

Selector

Register

Result

ADD

4
Store: Stage 3

IF: Instruction Fetch
ID: Instruction decoder
MEM: Memory access
WB: Write Back

Selector
PC
Add

4

Address
Instruction Memory
Instruction

Selectors

Register 1
Register 2
Registers

Read register 1
Read register 2
Read data 1
Read data 2
Write register
Write data

Sign
extended

Shift
Left

Add
ADD

Zero

ALU
Result

Address
Data Memory

Selector

Write data

Selector

Read data

Write Back
Store: Stage 4

IF: Instruction Fetch

ID: Instruction decoder
   register file read

EX: Execute /
    address calculate

MEM: Memory access

WB: Write Back

Selector

PC

Selector

Address

Instruction Memory

Instruction

Selector

4

Add

Sign extended

Shift Left

Zero

ALU

Result

Registers

Read register 1
Read data 1

Read register 2
Read data 2

Write register
Write data

Address

Read data

Data Memory

Write data

Address

Read data

Data Memory

Write data

Selector
Store: Stage 5

IF: Instruction Fetch
- Address
- Instruction Memory
- Instruction
- PC
- Selector

ID: Instruction decoder
- Read register 1
- Read register 2
- Registers
- Write register
- Read data 1
- Read data 2
- Write data

EX: Execute / address calculate
- Add
- Sign extended
- Shift Left
- ADD
- Zero
- ALU Result

MEM: Memory access
- Write data
- Data Memory
- Address

WB: Write Back
- Read data
- Selector
add
Add: Stage 1

IF: Instruction Fetch
- 4
- Address
- Instruction Memory
- Instruction

ID: Instruction decoder
- register file read
- Read register 1
- Read register 2
- Write register

EX: Execute / address calculate
- Sign extended
- Shift Left
- ADD
- Zero
- ALU Result
- Selector

MEM: Memory access
- Address
- Data Memory
- Read data
- Write data

WB: Write Back
- Selector
- Write data
Add: Stage 2

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute / address calculate

MEM: Memory access

WB: Write Back

Selector
PC
Add

4

Address
Instruction Memory
Instruction

Registers
Read register 1
Read register 2
Write register
Write data

Read data 1
Read data 2

Sign extended
Shift Left

ADD

Zero

ALU Result

Selector

Address
Read data
Data Memory
Write data

 selector

selector
Add: Stage 3

IF: Instruction Fetch
- Selector
- PC
- Address Memory
- Instruction Memory
- Instruction

ID: Instruction decoder
- Register file read
- Add
- Sign extended
- Shift Left
- Read register 1
- Read register 2
- Write register
- Read data 1
- Read data 2
- Write data

EX: Execute / address calculate
- ADD
- Zero
- ALU Result
- Selector
- Address Memory
- Data Memory
- Write data

MEM: Memory access
- Write Back

WB: Write Back
- Selector
Add: Stage 4

IF: Instruction Fetch

ID: Instruction decoder register file read

EX: Execute / address calculate

MEM: Memory access

WB: Write Back

4

Add

Address

Instruction Memory

Instruction

Selector

PC

Add

Sign extended

Shift Left

Zero

ALU Result

Address

Data Memory

Write data

Selector

Read register 1

Read register 2

Registers

Write register

Read data 1

Read data 2

Write data

Selector

Write data

Address

Read data
Add: Stage 5

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Selector
PC

Add

Address
Instruction
Memory

Instruction

Selector

4

Add

Sign
extended

Shift
Left

Zero

ALU
Result

Address
Read
data

Data
Memory
Write
data

Specify your text here.
write to register
Which Register?

IF: Instruction Fetch
- 4
- Add
- PC
- Selector
- Address Memory
- Instruction
- Instruction

ID: Instruction decoder
- register file read
- Read register 1
- Read register 2
- Write register
- Write data
- ALU Add Instruction Memory Address Instruction Registers Read register 1 Read register 2 Write register Write data

EX: Execute /
- address calculate
- Sign extended
- Shift Left
- ADD
- Zero
- ALU Result
- Selector
- Address Data Memory Write data

MEM: Memory access
- Zero
- ALU
- Result
- Selector
- Read data

WB: Write Back
- Selector
- Address
- Read data
- Data Memory
- Write data
Problem

• Write register
  – decoded in stage 2
  – used in stage 5

• Identity of register has to be passed along
Data Path for Write Register
pipelined control
Pipelined Control

- At each stage, information from instruction is needed
  - which ALU operation to execute
  - which memory address to consult
  - which register to write to

- This control information has to be passed through stages
Pipelined Control

IF  ID  EX  MEM  WB
    WB  WB
    M  M
    WB  WB
    WB

Control
Control Flags

IF: Instruction Fetch
ID: Instruction decoder register file read
EX: Execute / address calculate
MEM: Memory access
WB: Write Back

Selector
PC

Add

Selector

Instruction Memory
Instruction

4

Address

Instruction

Add

Read register 1
Read data 1
Read register 2
Read data 2
Write register
Write data
Register Write

Sign extended
Shift Left

ADD

Zero

ALU

Address

Instruction

Memory

Data

Memory

R/W

Selector

Branch (req. add. logic)

ALU Source

ALU Operation

Selector

Acknowledgement