Laundry Analogy

6pm 7pm 8pm 9pm 10pm 11pm

Task A

Task B

Task C

Task D
Laundry Pipelined

6pm  7pm  8pm  9pm  10pm  11pm

Task A
Task B
Task C
Task D
Speed-up

• Theoretical speed-up: 3 times

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

• Many tasks $\rightarrow$ 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.</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>Brand (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)
• Theoretical speed-up: 4 times

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

• Many tasks → 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
• 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>
<tr>
<td>conflict</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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

  ⇒ 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
```

```plaintext
IF
ID
EX
MEM
WB
```
Add and Subtract

IF
add $s0,$t0,$t1

ID
MEM
WB

EX

ID
MEM
WB

EX

IF
sub $t0,$s0,$t3

• 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

lw $t1, 0($t0)  lw $t1, 0($t0)
lw $t2, 4($t0)  lw $t2, 4($t0)
add $t3, $t1, $t2  lw $t4, 8($t0)
sw $t3, 12($t0)  add $t3, $t1, $t2
lw $t4, 8($t0)  sw $t3, 12($t0)
add $t5, $t1, $t4  add $t5, $t1, $t4
sw $t5, 16($t0)  sw $t5, 16($t0)

• 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
  \[ \rightarrow \text{full speed if correct} \]

- More sophisticated
  - keep record of branch taken or not
  - make prediction based on history
pipelined data path
load
Load: Stage 1

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

Selector
PC

Address Memory
Instruction

Add

4

Instruction

Selector

Write
data

Write
register

Read
register 1

Read
register 2

Read
data 1

Read
data 2

ALU

Add

Instruction

Memory
Address
Instruction

Zero

Shift Left

Register

Sign extended

ADD

EX: Execute /
address calculate

Write Back

Selector

Address

Selector

Read data

Data Memory
Write data

Zero

Result

Address

Selector
Load: Stage 2

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

Selector
PC
Add
Address
Instruction Memory
Instruction

Selector
Address
Read register 1
Read register 2
Write register
Write data

Selector
Address
Read data

Register
Add
Instruction
Memory
Address
Instruction
Read register 1
Read register 2
Write register
Write data

Sign extended
Shift Left
ADD
Zero
ALU Result

Address
Read data
Data Memory
Write data

Selector

**Load: Stage 4**

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

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

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

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

**WB:** Write Back
- Selector
Load: Stage 5

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

Selector
PC

Address
Instruction Memory
Instruction

Add

4

Selector

Zero

Write data

Selector

ALU Result

Read data

Address

Write data

Read register 1

Read register 2

Write register

Read data 1

Read data 2

Register

Write data

Sign extended

Shift Left

ADD

Address

Write data

Data Memory

Selector
store
Store: Stage 2

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

Selector
Add
Address
Instruction Memory
Instruction
Register 1
Register 2
Read data 1
Read data 2
Write register
Write data

Sign extended
Shift Left
ADD
Zero
ALU
Result
Address
Read data
Data Memory
Write data

Selector
Selector
4
Store: Stage 3

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

Register
Address
Read register 1
Read data 1

Read register 2
Read data 2

Write register 2
Write data

Select

Sign
extended

Shift
Left

Zero

ALU
Result

Add

Selector

Address
Read data

Data Memory
Write data

Selector
Store: Stage 5

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

Selector
PC
Add
Address
Instruction
Memory

Instruction decoder
register file read
MEM: Memory access

Selector
Register
Add

Select

Sign extended
Shift Left
Zero

ALU

Read register 1
Read register 2
Write register
Write data

Register

Read data 1
Read data 2
Write data

Zero

Address
Read data
Data Memory
Write data

Zero

Selector
add
Add: Stage 1

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

Selector
PC
Add
Address
Instruction Memory
Instruction
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Write data

Sign extended
Shift Left
Zero
ALU Result

Address
Read data
Data Memory
Write data
Selector
**Add: Stage 2**

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

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

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

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

**WB:** Write Back
- Selector

---

Philipp Koehn
Computer Systems Fundamentals: Pipelining
26 March 2018
Add: Stage 3

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

Selector

Read register 1

Read data 1

Write register

Read data 2

Write data

Address

Read data

Data Memory

Write data
Add: Stage 4

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

Selector → PC

4 → Add

Selector

Instruction Memory
Instruction

Address

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

Selector

Sign extended
Shift Left

ADD

Zero

ALU Result

Selector

Address

Read data

Data Memory
Write data

Add: Stage 4

Instruction Fetch
Instruction decoder
Execute
Memory access
Write Back

Register file read

Instruction decoder

Register file read

Address calculate

Write Back

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

Sign extended
Shift Left

ADD

Zero

ALU Result

Selector

Address

Read data

Data Memory
Write data

Add: Stage 4

Instruction Fetch
Instruction decoder
Execute
Memory access
Write Back

Register file read

Instruction decoder

Register file read

Address calculate

Write Back

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

Sign extended
Shift Left

ADD

Zero

ALU Result

Selector

Address

Read data

Data Memory
Write data
Add: Stage 5

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Selector

Add

4

Address

Instruction Memory

Instruction

PC

Read register 1
Read register 2
Write register
Write data

Read data 1
Read data 2

Sign extended

Shift Left

ADD

Zero

ALU

Selector

Result

Address

Data Memory

Write data

Select

 selectors:
write to register
Which Register?

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

Selector
PC

Add

Address
Instruction
Memory

Instruction decoder
register file read
address calculate

Selector

ADD

Shift Left

Zero

ALU
Result

Selector

Read register 1
Read register 2
Write register
Write data

Read data 1
Read data 2

Register

PC

Selector

Address
Data
Write data

Read data

Selector

Write Back
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