Pipelining

Philipp Koehn

24 October 2016
Laundry Pipelined

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

Task A
- Wash
- Dry
- Fold

Task B
- Wash
- Dry
- Fold

Task C
- Wash
- Dry
- Fold

Task D
- Wash
- Dry
- 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 $\rightarrow$ 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)

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: $800\text{ps} + 800\text{ps} + 800\text{ps} = 2400\text{ps}$
  – 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
Hazard

- 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></td>
<td>FETCH</td>
<td>DECODE</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td>ALU</td>
</tr>
<tr>
<td>i2</td>
<td></td>
<td>FETCH</td>
<td>DECODE</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td>ALU</td>
</tr>
</tbody>
</table>

  *conflict*

- **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

  \[\Rightarrow\] 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

Figure shows the pipeline stages with a timeline from 200 to 1000 and an instruction `add $s0,$t0,$t1`.
Add and Subtract

- Add wiring to circuit to directly connect output of ALU for next instruction
• Add wiring from memory lookup to ALU

• Still 1 cycle unused: "pipeline stall" or "bubble"
Reorder Code

• Code with data hazard

```assembly
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

\[
\begin{align*}
\text{add } & $s0, $t0, $t1 \\
\text{beq } & $s0, $s1, ff40 \\
\text{sub } & $t0, $s0, $t3
\end{align*}
\]

→ 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
Add
PC
Address
Instruction Memory
Instruction

ID: Instruction decoder
register file read

Sign extended
Shift Left
ADD
Zero
ALU
Result

Selector

Instruction Fetch

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

Write register
Write data

Address
Data Memory
Read data
Write data
Pipelined Datapath

IF: Instruction Fetch
- Address
- Instruction Memory
- Instruction

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

EX: Execute / address calculate
- ADD
- Zero
- ALU
- Result

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

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

Selector
PC

4

Add

Selector

Instruction Fetch
Instruction decoder
register file read
Execute / address calculate
Memory access
Write Back
load
Load: Stage 1

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

Select 4
Add

Instruction Memory
Instruction

Address

Register 1
Register 2

Read
Write

Read
Write

Read
Write

Write

Data

Add

Sign extended

Shift Left

Zero

Result

Selector

Address

Selector

Address

Philipp Koehn Computer Systems Foundamental: Pipelining 24 October 2016
Load: Stage 3

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

Selector
PC
Address
Instruction Memory
Instruction

4
Add

Sign extended
Shift Left

Zero

ALU Result

Selectors

Write register
Read data 2
Write data
Read register 2

Write register
Read data 1
Read register 1

Add

Zero

Address
Read data
Data Memory
Write data
Load: Stage 4

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

Selector
PC

Add

4

Address Memory
Instruction

Instruction decoder
register file read

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

Write register
Write data

Sign extended
Shift Left

ADD

Zero

ALU Result

Selector

Address Memory
Data Memory
Read data
Write data

Selector

Philipp Koehn
Computer Systems Foundamental: Pipelining
24 October 2016
store
Store: Stage 3

IF: Instruction Fetch

ID: Instruction decoder
- register file read

EX: Execute / address calculate

MEM: Memory access

WB: Write Back

- Selector
- PC
- Address
- Instruction Memory
- Instruction
- Add

- 4
- Sign extended
- Shift Left
- Zero
- ALU Result

- Write
- Register
- Read
- Register
- Data

- Selector
- Address
- Read
- Data
- Memory
- Write
- Data

Selector

Instruction Fetch

Instruction decoder

Execute / address calculate

Memory access

Write Back
Store: Stage 4

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

Selector
PC
Add
Address
Instruction Memory
Instruction

Selector

4

ADD
Zero
ALU Result
Selector

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

Selector
Write register
Read data
Write data

Selector

Data Memory
Address
Read data
Write data

Selector
Store: Stage 5
add
Add: Stage 1

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

Select 4
Add

Address
Instruction Memory
Instruction

Selector
PC

Add

Sign extended
Shift Left
ADD

ALU

Selector

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

Zero

Address
Data Memory
Write data

Selector

36
Add: Stage 2

IF: Instruction Fetch

ID: Instruction decoder
   register file read

EX: Execute /
    address calculate

MEM: Memory access

WB: Write Back

Selector
PC

Address
Instruction
Memory

Instruction

Add

Selector

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

Register

Write data

Shift Left

ADD

Zero

ALU Result

Selector

Address
Read data
Data Memory
Write data

Selector

Philipp Koehn
Computer Systems Foundamental: Pipelining
24 October 2016
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

4

Address
Instruction Memory
Instruction

Read register 1
Read register 2

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

Zero

Shift Left

ADD

Selector

Address
Data Memory
Read data
Write data

Selector
Add: Stage 4

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Selector
PC

4

Add

Address

Instruction Memory

Instruction

Sign
extended

Shift
Left

ADD

Zero

ALU
Result

Selector

Address
Read data

Data Memory

Write data

Read register 1
Read data 1

Read register 2
Read data 2

Write register
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
PC

4

Add

Address

Instruction Memory

Instruction

Read register 1
Read register 2
Write register
Write data

Registers

Read data 1
Read data 2

Sign extended

Shift Left

ADD

Zero

ALU
Result

Selector

Address
Read data
Data Memory

Write data

Sector
write to register
Which Register?

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

ID: Instruction decoder register file read
- 4
- Add
- Write register
- Write data
- Read register 1
- Read data 1
- Read register 2
- Read data 2

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

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

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

Selector

Instruction Fetch
Instruction decoder
Register file read
Execute
Address calculate
Memory access
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

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

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

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

MEM: Memory access
- Write data
- Selector

WB: Write Back
- Address
- Data Memory
- Write data
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

Control

IF

ID

EX

MEM

WB
Control Flags