Pipelining the MIPS Datapath
Today’s lecture

- Pipeline implementation
  - Single-cycle Datapath
  - Pipelining performance
  - Pipelined datapath
  - Example
We have built a single-cycle implementation of a subset of the MIPS-based instruction set

- We have assumed that instructions execute in the same amount of time; this determines the clock cycle time.
- We have implemented the datapath and the control unit.
Refresher: The Full Single-cycle Datapath
We will use a simplified implementation of MIPS to create a pipelined version

Arithmetic:    add    sub    and    or    slt

Data Transfer:    lw    sw

Control:        beq
Worst-case delay from register-read to register-write determines clock speed
How fast can we clock a pipelined datapath?

2ns 2ns 2ns 1ns 1ns
Break datapath into 5 stages

IF

Read Instruction address [31-0]

Instruction memory

ID

RegWrite

I [25 - 21]

I [20 - 16]

I [15 - 11]

I [15 - 0]

RegDst

0 Mul

1

EXE

ALU

Zero

Result

0 Mul

1

ALUSrc

MEM

MemWrite

MemRead

MemToReg

WB

1 Mul

0

Clock cycle

1 2 3 4 5 6 7 8 9

lw $t0, 4($sp) IF ID EX MEM WB
lw $t1, 8($sp) IF ID EX MEM WB
lw $t2, 12($sp) IF ID EX MEM WB
lw $t3, 16($sp) IF ID EX MEM WB
lw $t4, 20($sp) IF ID EX MEM WB
Ideal pipeline performance is **time to fill the pipeline + one cycle per instruction**

- How long for N instructions on pipelined architecture?
- How long for N instructions on single-cycle (8ns clock period)?
- How much faster is pipelining for N=1000?
Pipelining improves throughput at the cost of increased latency
Some instructions do not require all five stages, can we skip stages?

- Example: R-type instructions only require 4 stages: IF, ID, EX, and WB
  - We don’t need the MEM stage
- What happens if we try to pipeline loads with R-type instructions?

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $sp, $sp, -4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $v0, $a0, $a1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $t0, 4($sp)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $s0, $s1, $s2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $t1, 8($sp)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Trying to use the single stage for multiple instructions creates a structural hazard

- Each functional unit can only be used once per instruction
- Each functional unit must be used at the same stage for all instructions:
  - Load uses Register File’s Write Port during its 5th stage
  - R-type uses Register File’s Write Port during its 4th stage

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add $sp, $sp, -4</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>sub $v0, $a0, $a1</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lw $t0, 4($sp)</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td><strong>MEM</strong></td>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>or $s0, $s1, $s2</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lw $t1, 8($sp)</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td><strong>MEM</strong></td>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Insert NOP stages to avoid structural hazards

- All instructions take 5 cycles with the same stages in the same order
- Some stages will do nothing for some instructions

R-type

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add $sp, $sp, -4</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>NOP</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>sub $v0, $a0, $a1</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>NOP</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lw $t0, 4($sp)</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>or $s0, $s1, $s2</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>NOP</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lw $t1, 8($sp)</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Stores and Branches have NOP stages, too...

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>store</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>NOP</td>
</tr>
<tr>
<td><strong>branch</strong></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>NOP</td>
<td>NOP</td>
</tr>
</tbody>
</table>
Single-cycle datapath rearranged to align with pipeline stages
Add pipeline registers in between stages

- There’s a lot of information to save, however. We’ll simplify our diagrams by drawing just one big **pipeline register** between each stage.
- The registers are named for the stages they connect.
  
  IF/ID    ID/EX    EX/MEM    MEM/WB

- No register is needed after the WB stage, because after WB the instruction is done.
Paths from register-read to register-write are now shorter
Data values required in later stages must be **propagated forward** through the pipeline registers.

- **Example**
  - Destination register \((r_d)\) is determined during the *first* stage (IF)
  - We store into the destination register during the *fifth* stage (WB)
  - \(r_d\) must be passed through all of the pipeline stages
Note – We cannot keep values like destination register in the “instruction register”
Control signals are generated in the decode stage and are propagated across stages.

Group control signals based on the stages they are used in.
Categorize control signals by the pipeline stage that uses them

<table>
<thead>
<tr>
<th>Stage</th>
<th>Control signals needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX</td>
<td>ALUSrc</td>
</tr>
<tr>
<td></td>
<td>ALUOp</td>
</tr>
<tr>
<td></td>
<td>RegDst</td>
</tr>
<tr>
<td></td>
<td>PCSrc</td>
</tr>
<tr>
<td>MEM</td>
<td>MemRead</td>
</tr>
<tr>
<td></td>
<td>MemWrite</td>
</tr>
<tr>
<td>WB</td>
<td>RegWrite</td>
</tr>
<tr>
<td></td>
<td>MemToReg</td>
</tr>
</tbody>
</table>
The pipeline registers and program counter update every clock cycle, so they do not have write enable controls.
## An example execution sequence

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>lw</td>
<td>$8, 4($29)</td>
</tr>
<tr>
<td>1004</td>
<td>sub</td>
<td>$2, $4, $5</td>
</tr>
<tr>
<td>1008</td>
<td>and</td>
<td>$9, $10, $11</td>
</tr>
<tr>
<td>1012</td>
<td>or</td>
<td>$16, $17, $18</td>
</tr>
<tr>
<td>1016</td>
<td>add</td>
<td>$13, $14, $0</td>
</tr>
</tbody>
</table>

### ASSUMPTIONS

### CONVENTIONS
- X indicates values that are not important, Example: Imm16 for R-type.
- Question marks ??? indicate values we do not know, usually resulting from instructions coming before and after the ones in our example.
Cycle 1 (filling)

IF: lw $8, 4($29)

Mem: ???
WB: ???

Read address [31-0]
Instruction memory

Shift left 2
ALUSrc (?)

ADD

1
0

1000

RegWrite (?)
RegWrite data
Write register 2
Write register 1

ALU Zero
Result

RegDst (?)

Write data

ALUSrc (?)

Sign extend

Read register 2
Read register 1

MEM: ???

ID: ???

EX: ???

MEM/WB

MEM/ WB

1

PCSrc

4

Add

PC

ID/EX

M

EX/ MEM

Write data

WRITE

MEM

MemWrite (?)

MemRead (?)

MEM/ WB

1

0

MEMToReg (?)

RegWrite (?)

ALUSrc (?)

 ALUOp (???)

ZeroALU

Add

Shift left 2


ID/EX

M

EX/ MEM

M

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB

WB


Cycle 3

IF: and $9, $10, $11
ID: sub $2, $4, $5
EX: lw $8, 4($29)
MEM: ???
WB: ???

Read Instruction address [31-0]

Instruction memory

RegWrite (?)

Read register 1
Read data 1
Write register
Write data

Shift left 2

ALU Zero
Result

ALUSrc (___)

RegDst (___)

MemRead (?)

MemWrite (?)

MemToReg

RegWrite (?)

Add

PCSrc

Add

Shift

ZeroALU

ALUOp (___)

MemWrite (?)

MemRead (?)

Write data

Write data 2

Sign extend

Write register 2

Write data

PC

1008

1012

1

0

4

5

X

X

2
**Cycle 4**

**Instruction Memory**
- **Read** instruction address [31-0]
- **PC**: 1012
- Instruction memory

**Control**
- **RegWrite (?)**
- **Write register**
- **Write data**
- **RegDst (1)**
- **Sign extend**
- **ALUZero Result**
- **ALUSrc (0)**
- **ALUOp (sub)**

**Register File**
- **Read register 1**
- **Read register 2**
- **Write data**

**ALU**
- **Shift left 2**
- **Add**

**Memory**
- **MemWrite (___)**
- **MemRead (___)**

**Register File**
- **MemToReg (?)**
- **Write data**
- **Read data**

**Control Flow**
- **IF: or $16, $17, $18**
- **ID: and $9, $10, $11**
- **EX: sub $2, $4, $5**
- **MEM: lw $8, 4($29)**
- **WB: ??]**

**Memory to Register**
- **MemWrite (___)**
- **MemRead (___)**

**Register File**
- **RegWrite (?)**
- **Write data**
- **Read data 1**
- **Read data 2**

**Instruction Memory**
- **Read address [31-0]**

**IF/ID**
- **Add**
- **Add**

**ID/EX**
- **Shift left 2**
- **Add**

**EX/MEM**
- **ALUZero Result**
- **ALUSrc (0)**

**MEM/WB**
- **MemWrite (___)**
- **MemRead (___)**

**Register File**
- **RegWrite (?)**
- **Write data**
- **Read data 1**
- **Read data 2**

**Control**
- **RegWrite (?)**
- **Write register**
- **Write data**
- **RegDst (1)**
- **Sign extend**
- **ALUZero Result**
- **ALUSrc (0)**
- **ALUOp (sub)**

**Memory**
- **MemWrite (___)**
- **MemRead (___)**

**Register File**
- **MemToReg (?)**
- **Write data**
- **Read data**

**ALU**
- **Shift left 2**
- **Add**

**Memory**
- **MemWrite (___)**
- **MemRead (___)**

**Register File**
- **RegWrite (?)**
- **Write data**
- **Read data 1**
- **Read data 2**
Cycle 5 (full)

IF: add $13, $14, $0
ID: or $16, $17, $18
EX: and $9, $10, $11
MEM: sub $2, $4, $5
WB: lw $8, 4($29)
Cycle 6 (emptying)

IF: ???
ID: add $13, $14, $0
EX: or $16, $17, $18
MEM: and $9, $10, $11
WB: sub $2, $4, $5
Cycle 7

IF: ???

ID: ???

EX: add $13, $14, $0

MEM: or $16, $17, $18

WB: and $9, $10, $11

Read address [31-0]

Instruction memory

MemWrite (0)

MemRead (0)

Shift left 2

Add

ALUSrc (0)

Result

ALUOp (add)

Address

Data memory

Write data

Read data

RegWrite (1)

Read register 1
data 1

Read register 2
data 2

Write register

Write data

Registers

RegDst (1)

X

1

0

13

119

114

118

110

110

9

110

???

???

???

???

???

???

???

???

???

???

???

???

???

???

???
Cycle 9

IF: ??? ID: ??? EX: ??? MEM: ???

ID/EX

Control

Add

Shift left 2

ALU Zero

RESULT

MemWrite (?)

MemToReg (0)

IF/ID

Read

Instruction address [31-0]

Instruction memory

IF/ID

RegWrite (1)

Read register 1

Read data 1

Write register

Write data

Read register 2

Read data 2

Write Registers

ID/EX

ALUSrc (?)

ALUOp (?)

Data memory

Data

Read data

Write data

MemRead (?)

MEM/WB

MEM/EX

EX/MEM

EX/ID

Add

RegDst (?)

Sign extend

WB: add $13, $14, $0
Things to notice from the last nine slides

- Instruction executions overlap
- Each functional unit is used by a *different* instruction in each cycle.
- In clock cycle 5, all of the hardware units are used (the pipeline is full). This is the ideal situation, and what makes pipelined processors so fast
- Similar example in the book available at the end of Section 6.3.
MIPs ISA makes pipelining “easy”

- Instruction formats are the same length and uniform
- Addressing modes are simple
- Each instruction takes only one cycle
Note how everything goes left to right, except ... Next time: We will discuss Data Hazards.