Basic Processor Design

Hung-Wei Tseng
Outline

- Single-cycle processor
- Pipelining
- The 5-stage MIPS Pipeline processor
- Pipeline Hazards
- Structural Hazards
- Control Hazards
- Dynamic Branch Predictions
The "life" of a dynamic instruction?

- Instruction Fetch (IF)
  - Fetch the instruction pointed by PC from memory
- Instruction Decode (ID)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (EX)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (MEM) — Read/write data memory
- Write Back (WB) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4
Single-cycle processor
Single-cycle processor

Instruction Fetch

Instruction Memory

Instruction Address [31:0] Instruction [31:0]

Instruction Decode, prepare operands

Register File

Read Reg 1 Read Data 1

Read Reg 2 Read Data 2

Write Reg Write Data

Control

RegDst MemToReg Branch MemoryRead MemoryWrite ALUOp RegWrite

Next PC

ALU

Data Memory

Is zero?

ALU Ctrl.

Write Back

Write Data[31:0]

Data Memory

Data Address [31:0]

Write Data[31:0]

Write Back

Next PC

shift left 2

Instruction Fetch

Instruction Decode, prepare operands

Instruction Memory

Instruction Address [31:0] Instruction [31:0]

Instruction Decode, prepare operands

Register File

Read Reg 1 Read Data 1

Read Reg 2 Read Data 2

Write Reg Write Data

ALU

Data Memory

Is zero?

ALU Ctrl.

Write Back

Next PC

Instruction fetch
Pipelining
Pipelining

- Different parts of the processor works on different instructions simultaneously
- A clock signal controls and synchronize the beginning and the end of each part of the work
- A pipeline register between different parts of the processor to keep intermediate results necessary for the upcoming work
Pipelining
After this point, we are completing an instruction each cycle!

All hardware parts are in use
Cycle time of a pipeline processor

- Critical path is the longest possible delay between two registers in a design.
- The critical path sets the cycle time, since the cycle time must be long enough for a signal to traverse the critical path.
- Lengthening or shortening non-critical paths does not change performance.
- Ideally, all paths are about the same length.
The 5-stage MIPS Pipeline Processor
Single-cycle processor
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9, $10, $11
sw  $1, 0($12)
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9, $10, $11
sw $1, 0($12)
After this point, we are completing an instruction each cycle!
Each instruction has to go through all 5 pipeline stages: IF, ID, EXE, MEM, WB in order — only valid if it’s single-issue, MIPS 5-stage pipeline

An instruction can enter the next pipeline stage in the next cycle if

- No other instruction is occupying the next stage
- This instruction has completed its own work in the current stage
- The next stage has all its inputs ready and it can retrieve those inputs

Fetch a new instruction only if

- We know the next PC to fetch
- We can predict the next PC

Flush an instruction if the branch resolution says it’s mis-predicted.
Pipeline hazards
Three pipeline hazards

• Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
• Control hazards — the PC can be changed by an instruction in the pipeline
• Data hazards — an instruction depending on a the result that’s not yet generated or propagated when the instruction needs that
Can we get them right?

- Given a simple pipelined MIPS processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
</tr>
<tr>
<td>b</td>
<td>lw $4, 0($1)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
</tr>
<tr>
<td>c</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
<td>sub $9, $1, $10</td>
<td>sub $9, $10, $11</td>
</tr>
<tr>
<td>d</td>
<td>sub $9, $10, $11</td>
<td>sub $9, $1, $10</td>
<td>sub $1, 0($12)</td>
<td>sub $9, $10, $11</td>
</tr>
<tr>
<td>e</td>
<td>sw $1, 0($12)</td>
<td>sw $11, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
</tr>
</tbody>
</table>

A. 0

B. 1

C. 2

D. 3

E. 4

- b cannot get x1 produced by a before WB
- both a and d are accessing x1 at the 5th cycle
- We don’t know if d & e will be executed or not until c finishes

Data Hazard

Structural Hazard

Control Hazard
Structural Hazards
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: insert no-ops (e.g, add $0, $0, $0) between them
- Drawback
  - If the number of pipeline stages changes, the code won’t work
  - Slow

```
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
add $0, $0, $0
sub $9, $1, $10
sw $11, 0($12)
```
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: stall the later instruction, allowing the write to present the change in the register and the later can get the desired value
- Drawback: slow

```
add $1, $2, $3  
lw  $4, 0($5)  
sub $6, $7, $8  
sub $9, $1, $10  
sw  $11, 0($12)
```
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Better solution: write early, read late
  - Writes occur at the clock edge and complete long enough before the end of the clock cycle.
  - This leaves enough time for outputs to settle for reads
  - The revised register file is the default one from now!

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9, $1, $10
sw  $11, 0($12)
```
Structural Hazards

- What pair of instructions will be problematic if we allow ALU instructions to skip the “MEM” stage?

- a: lw $1, 0($2)
- b: add $3, $4, $5
- c: sub $6, $7, $8
- d: sub $9, $10, $11
- e: sw $1, 0($12)

A. a & b
B. a & c
C. b & e
D. c & e
E. None
Structural Hazards

- Stall can address the issue — but slow
- Compiler optimization — what if the hardware design changes?
- Improve the pipeline unit design to allow parallel execution