Processor Design (II)

Hung-Wei Tseng
Single cycle processor
Performance of a single-cycle processor

• How many of the following statements about a single-cycle processor is correct?
  ① The CPI of a single-cycle processor is always 1
  ② If the single-cycle implements MIPS ISA, the memory instruction will determine the cycle time
  ③ Hardware elements are mostly idle during a cycle
  ④ We can always reduce the cycle time of a single-cycle processor by supporting fewer instructions — Only if this instruction is the most time-critical one

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Pipelining

- Break up the logic with “pipeline registers” into pipeline stages
  - These registers only changes their output at the triggered edge cycle
- Each stage can act on different instruction/data
- States/Control signals of instructions are hold in pipeline registers
The processor can complete 1 instruction each cycle

CPI == 1 if everything works perfectly!
Single-cycle v.s. pipeline v.s.
Limitations of pipelining

- How many of the following descriptions about pipelining is correct?
  - You can always divide stages into short stages with latches to improve performance
    - Only if this stage is the most time-critical one
  - Pipeline registers incur overhead for each pipeline stage
  - The latency of executing an instruction in a pipeline processor is longer than a single-cycle processor
    - You have pipeline registers and each stage needs to be equally long
  - The throughput of a pipeline processor is usually better than a single-cycle processor

A. 0
B. 1
C. 2
D. 3
E. 4
Outline

• Pipeline hazards
Single cycle processor
5-stage pipeline processor
5-stage pipeline processor
5-stage pipeline processor

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
```
5-stage pipeline processor

- **add $1, $2, $3**
- **lw  $4, 0($5)**
- **sub $6, $7, $8**
- **sub $9, $10, $11**
- **sw  $1, 0($12)**
5-stage pipeline 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)
5-stage pipeline processor

add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
• Use symbols to represent the physical resources with the abbreviations for pipeline stages.
  • IF, ID, EXE, MEM, WB
• The horizontal axis represents the timeline, and the vertical axis represents the instruction stream
• Example:

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
```
Given the current 5-stage pipeline, how many of the following MIPS code can work correctly (i.e. generate the same result as a single-cycle processor)?

<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>bne $0, $7, L</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d</td>
<td>sub $9,$10,$11</td>
<td>sub $9,$1,$10</td>
<td>sub $9,$10,$11</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
Can we get the right result?

Given the current 5-stage pipeline, how many of the following MIPS code can work correctly (i.e. generate the same result as a single-cycle processor)?

<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>bne $0, $7, L</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d:</td>
<td>sub $9,$10,$11</td>
<td>sub $9, $1, $10</td>
<td>sub $9,$10,$11</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  

<table>
<thead>
<tr>
<th>Response</th>
<th>Vote %</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4%</td>
</tr>
<tr>
<td>B</td>
<td>8%</td>
</tr>
<tr>
<td>C</td>
<td>35%</td>
</tr>
<tr>
<td>D</td>
<td>54%</td>
</tr>
<tr>
<td>E</td>
<td>0%</td>
</tr>
</tbody>
</table>

20
Given the current 5-stage pipeline, how many of the following MIPS code can work correctly (i.e. generate the same result as a single-cycle processor)?

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>b:</td>
<td>lw $4, 0($1)</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 $6, $7, $8</td>
</tr>
<tr>
<td>d:</td>
<td>sub $9,$10,$11</td>
<td>sub $9, $1, $10</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>
</tr>
</tbody>
</table>

- b cannot get $1 produced by a before WB
- both a and d are accessing $1 at 5th cycle
- We don’t know if d & e will be executed or not until c finishes

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Pipeline hazards
Pipeline hazards

• Even though we perfectly divide pipeline stages, it’s still hard to achieve CPI == 1.

• Pipeline hazards:
  • Structural hazard
    • The hardware does not allow two pipeline stages to work concurrently
  • Data hazard
    • A later instruction in a pipeline stage depends on the outcome of an earlier instruction in the pipeline
  • Control hazard
    • The processor is not clear about what’s the next instruction to fetch
Can we get the right result?

- Given the current 5-stage pipeline,

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EXE</th>
<th>MEM</th>
<th>WB</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 $6, $7, $8</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d:</td>
<td>sub $9,$10,$11</td>
<td>sub $9, $1, $10</td>
<td>sub $9,$10,$11</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>

- Data hazard: b cannot get $1 produced by a before WB
- Structural hazard: both a and d are accessing $1 at 5th cycle
- Control hazard: We don’t know if d & e will be executed or not

how many of the following MIPS code can work correctly?
Structural hazard
What just happened here is problematic if we change one of the source register of the 2nd sub instruction?

A. The register file is trying to read and write at the same cycle
B. The ALU and data memory are both active at the same cycle
C. A value is used before it’s produced
D. Both A and B
E. Both A and C
The hardware cannot support the combination of instructions that we want to execute at the same cycle.

The original pipeline incurs structural hazard when two instructions competing the same register.

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,$10, $1
sw $1, 0($12)
```
What pair of instructions will be problematic if we allow R-type instructions to skip the “MEM” stage?

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

- The design of hardware causes structural hazard
- We need to modify the hardware design to avoid structural hazard
Data hazard
What just happened here is problematic for the following instructions in our current pipeline?

A. The register file and memory are both active at the same cycle
B. The ALU and data memory are both active at the same cycle
C. A value is used before it’s produced
D. Both A and B
E. Both A and C
Data hazard

- When an instruction in the pipeline needs a value that is not available
- Data dependences
  - The output of an instruction is the input of a later instruction
  - May result in data hazard if the later instruction that consumes the result is still in the pipeline
Data dependences

• How many pairs of data dependences are there in the following code?
  add $1, $2, $3
  lw  $4, 0($1)
  sub $5, $2, $4
  sub $1, $3, $1
  sw  $1, 0($5)

A. 1
B. 2
C. 3
D. 4
E. 5
How many pairs of data dependences are there in the following code?

add $1, $2, $3
lw $4, 0($1)
sub $5, $2, $4
sub $1, $3, $1
sw $1, 0($5)

A. 1
B. 2
C. 3
D. 4
E. 5

Data dependences

- How many pairs of data dependences are there in the following code?

No every “data dependency” will lead to “data hazards”. 

E. 5
Sol. of data hazard I: Stall

- When the source operand of an instruction is not ready, stall the pipeline
  - Suspend the instruction and the following instruction
  - Allow the previous instructions to proceed
  - This introduces a pipeline bubble: a bubble does nothing, propagate through the pipeline like a nop instruction

- How to stall the pipeline?
  - Disable the PC update
  - Disable the pipeline registers on the earlier pipeline stages
  - When the stall is over, re-enable the pipeline registers, PC updates
Hazard detection & stall

Check if the destination register of EX == source register of the instruction in ID

Check if the destination register of MEM == source register of the instruction in ID

Insert a “noop” if we need to stall
Performance of stall

add $1, $2, $3
lw  $4, 0($1)
sub $5, $2, $4
sub $1, $3, $1
sw  $1, 0($5)

15 cycles! CPI == 3
(If there is no stall, CPI should be just 1!)
Sol. of data hazard II: Forwarding

- The result is available after EXE and MEM stage, but publicized in WB!
- The data is already there, we should use it right away!
- Also called bypassing

```
add $1, $2, $3
lw $4, 0($1)
sub $5, $2, $4
sub $1, $3, $1
sw $1, 0($5)
```

We can obtain the result here!
Sol. of data hazard II: Forwarding

- Take the values, where ever they are!

```
add $1, $2, $3
lw $4, 0($1)
sub $5, $2, $4
sub $1, $3, $1
sw $1, 0($5)
```

10 cycles! CPI == 2 (Not optimal, but much better!)
When can/should we forward data?

- If the instruction entering the EXE stage consumes a result from a previous instruction that is entering MEM stage or WB stage
  - A source of the instruction entering EXE stage is the destination of an instruction entering MEM/WB stage
  - The previous instruction must be an instruction that updates register file
5-stage pipeline processor
5-stage pipeline processor
There is still a case that we have to stall...

- Revisit the following code:

```
add $1, $2, $3
lw $4, 0($1)
sub $5, $2, $4
sub $1, $3, $1
sw $1, 0($5)
```

If the instruction entering EXE stage depends on a load instruction that does not finish its MEM stage yet, we have to stall!
The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

1. add $1, $2, $3
2. lw $4, 0($1)
3. sub $5, $2, $4
4. sub $1, $3, $1
5. sw $1, 0($5)

A. (1) & (2)
B. (2) & (3)
C. (3) & (4)
D. (4) & (5)
E. (3) & (5)
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}