Pipeline: Introduction

These slides are derived from:
CSCE430/830 Computer Architecture course by Prof. Hong Jiang and Dave Patterson ©UCB

Some figures and tables have been derived from:
Computer System Architecture by M. Morris Mano
Pipelining Outline

Introduction
Defining Pipelining
Pipelining Instructions

Hazards
  Structural hazards
  Data Hazards
  Control Hazards
What is Pipelining?

A way of speeding up execution of instructions

Key idea: overlap execution of multiple instructions
The Laundry Analogy

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- “Folder” takes 30 minutes
- “Stasher” takes 30 minutes to put clothes into drawers
If we do laundry sequentially...

Task Order

A
B
C
D

Time

6 PM 7 8 9 10 11 12 1 2 AM
To Pipeline, We Overlap Tasks

- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup
Pipelining a Digital System

- Key idea: break big computation up into pieces

Separate each piece with a pipeline register

- 200ps
- 200ps
- 200ps
- 200ps
Pipelining a Digital System

Why do this? Because it's faster for repeated computations

Non-pipelined: 1 operation finishes every 1ns

Pipelined: 1 operation finishes every 200ps
Comments about pipelining

Pipelining increases **throughput**, but not **latency**

Answer available every 200ps, BUT

- A single computation still takes 1ns

**Limitations:**

- Computations must be divisible into stage size
- Pipeline registers add overhead
• Suppose we need to perform multiply and add operation with a stream of numbers

\[ A_i \times B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7 \]

• Each subinstruction is implemented in a segment within the pipeline. Each segment has one or two registers and a combinational circuit

• The sub operations performed in each segment are as follows

  \[ R1 \leftarrow A_i, \quad R2 \leftarrow B_i \quad \text{Input } A_i \text{ and } B_i \]
  \[ R3 \leftarrow R1 \times R2, \quad R4 \leftarrow C_i \quad \text{Multiply and input } C_i \]
  \[ R5 \leftarrow R3 + R4 \quad \text{Add } C_i \text{ to product} \]
Example of Pipeline Processing
## Content of Registers in Pipeline

<table>
<thead>
<tr>
<th>Clock Pulse Number</th>
<th>Segment 1</th>
<th>Segment 2</th>
<th>Segment 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$R1$</td>
<td>$R2$</td>
<td>$R3$  $R4$</td>
</tr>
<tr>
<td>1</td>
<td>$A_1$</td>
<td>$B_1$</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>$A_2$</td>
<td>$B_2$</td>
<td>$A_1 \cdot B_1$</td>
</tr>
<tr>
<td>3</td>
<td>$A_3$</td>
<td>$B_3$</td>
<td>$A_2 \cdot B_2$</td>
</tr>
<tr>
<td>4</td>
<td>$A_4$</td>
<td>$B_4$</td>
<td>$A_3 \cdot B_3$</td>
</tr>
<tr>
<td>5</td>
<td>$A_5$</td>
<td>$B_5$</td>
<td>$A_4 \cdot B_4$</td>
</tr>
<tr>
<td>6</td>
<td>$A_6$</td>
<td>$B_6$</td>
<td>$A_5 \cdot B_5$</td>
</tr>
<tr>
<td>7</td>
<td>$A_7$</td>
<td>$B_7$</td>
<td>$A_6 \cdot B_6$</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>—</td>
<td>$A_7 \cdot B_7$</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
## Space Time Diagram of Pipeline

<table>
<thead>
<tr>
<th>Segment:</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>
<th>Clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Speedup

Speedup from pipeline

\[ S = \frac{\text{Average instruction time unpiplined}}{\text{Average instruction time pipelined}} \]

Consider a case for k-segment pipeline with a clock cycle time \( tp \) to execute \( n \) tasks. The first task \( T_1 \) requires a time equal to \( k*tp \) to complete its operation since there are \( k \) segments in pipeline. The remaining \( n-1 \) tasks emerge from the pipe at a rate of one task per clock cycle and they will be completed in \( k+n-1 \) clock cycles.

Next, to consider an unpipeline unit that performs the same operation and takes a time equal to \( tn \) to complete the task. The total time required for \( n \) tasks is \( n*tn \). The speed up of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio

\[ S = \frac{nt_n}{(k + n - 1)t_p} \]
Speedup

- As the number of tasks increase $n$ becomes much larger than $k-1$, and $k+n-1$ approaches the value of $n$. Under this condition, the speed up becomes

$$S = \frac{t_n}{t_p}$$

If we assume the the time taken to process the task is the same as in the pipeline and nonpipeline circuits, we will have $t_n = kt_p$

- The speedup then reduces to number of stages of pipeline

$$S = \frac{kt_p}{t_p} = k$$
Pipelining a Processor

• Recall the 5 steps in instruction execution:
  1. Instruction Fetch (IF)
  2. Instruction Decode and Register Read (ID)
  3. Execution operation or calculate address (EX)
  4. Memory access (MEM)
  5. Write result into register (WB)

• Review: Single-Cycle Processor
  – All 5 steps done in a single clock cycle
  – Dedicated hardware required for each step
Review - Single-Cycle Processor
The Basic Pipeline For MIPS

Cycle 1
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 2
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 3
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 4
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 5
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 6
- Ifetch
- Reg
- ALU
- DMem
- Reg

Cycle 7
- Ifetch
- Reg
- ALU
- DMem
- Reg

Instruction Order

Instruction Order
Basic Pipelined Processor
Single-Cycle vs. Pipelined Execution

**Non-Pipelined**

Instruction Order

1 w $1, 100($0)
1 w $2, 200($0)
1 w $3, 300($0)

**Pipelined**

Instruction Order

1 w $1, 100($0)
1 w $2, 200($0)
1 w $3, 300($0)
Comments about Pipelining

The good news
- Multiple instructions are being processed at same time
- This works because stages are isolated by registers
- Best case speedup of N

The bad news
- Instructions interfere with each other - hazards
  Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
  Example: instruction may require a result produced by an earlier instruction that is not yet complete
Pipeline Hazards

Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle

**Structural hazards**: two different instructions use same h/w in same cycle

**Data hazards**: Instruction depends on result of prior instruction still in the pipeline

**Control hazards**: Pipelining of branches & other instructions that change the PC