ECE 3057: Architecture, Concurrency and Energy in Computation
Summer 2019

Lecture 5: Performance & Pipelining

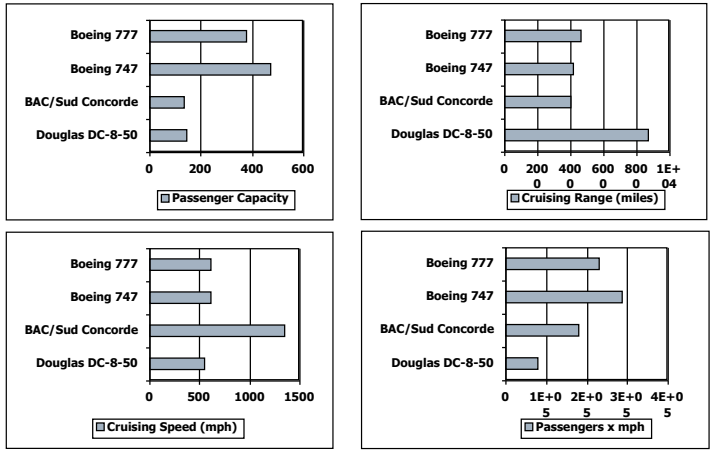
David Anderson

School of Electrical and Computer Engineering
Georgia Institute of Technology

Acknowledgment: Lecture slides adapted from GT ECE 3056 (S. Yalamanchilli & T. Krishna), MIT 6.823 (Arvind and J. Emer) and CMU 18-447 (O. Mutlu)

Defining Performance

- Example: which airplane has the best performance?
  - Depends on the metric you care about!



Throughput vs. Latency

Pilot + navigator – Mach 3,
NY to London in 2 hours
Average = 60 minutes/person

2200 crew + 6600 passengers – 22 knots
NY to London in 138 hours
Average = 1 minute/person

Defining Processor Performance

- What is the metric of performance?
  - Runtime
    - Time of completion
  - Throughput
    - Rate of completion
  - Energy-Efficiency
    - Runtime / Watt or Throughput / Watt
  - Thermal Efficiency
    - Temperature sensitivity
  - Cost-Efficiency
    - Performance / $

Processor Runtime

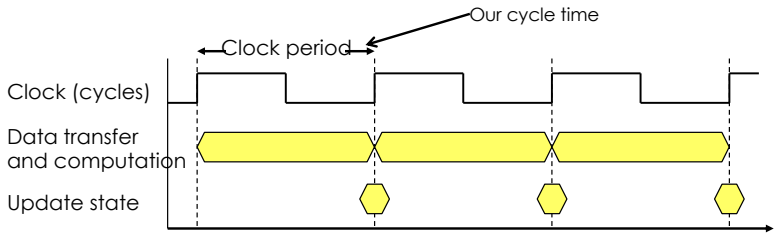
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

Instruction Count (IC) CPI Cycle Time

- **How does the compute stack affect runtime?**
  - Algorithm
    - IC
  - Programming language
    - IC
  - Compiler
    - IC
  - Instruction set architecture
    - IC, CPI
  - Microarchitecture
    - CPI, Cycle Time

Cycle-Time

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10–12s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×109Hz

Cycles Per Instruction

- Instruction Dependent in Modern Processors
  - Multiplication takes more time than addition
  - Floating point operations take longer than integer ones
  - Accessing memory takes (in general) more time than accessing registers
- *Note: changing the cycle time often changes the number of cycles required for various instructions*

Trading off CPI and Cycle Time

- Single-Cycle Datapath
  - CPI = 1, high cycle time
- Multi-Cycle Datapath
  - CPI = [2, 3, 4], low cycle time
- Pipelined Datapath (today)
  - CPI = 1.x, low cycle time

Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
  - How fast must Computer B clock be?

CPU Time A = 10 = IC x CPI_A x 1/(2E-9)
=> IC x CPI_A = 2E-8

CPU Time B = 6 = IC x CPI_B x $1/f_B$
=> f_B = (IC x CPI_B)/ 6 = 1.2 x (IC x CPI_A) / 6 = 4 GHz

Relative Performance

- "X is *n* times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution TimeB / Execution TimeA = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Program Execution time

Number of instruction classes

$$ExecutionTime = \left[\sum_{i=1}^{n} C_i \times CPI_i\right] \times cycle\_time$$

~= Instruction_count * CPI_{avg} * clock_cycle_time

algorithms/compiler architecture technology

$$CPI_{avg} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n}\left(CPI_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}}\right)$$

Relative frequency

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times CPI_A \times \text{Cycle Time}_A = I \times 2.0 \times 250ps = I \times 500ps$$

A is faster...

$$\text{CPU Time}_B = \text{Instruction Count} \times CPI_B \times \text{Cycle Time}_B = I \times 1.2 \times 500ps = I \times 600ps$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600ps}{I \times 500ps} = 1.2$$

...by this much

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

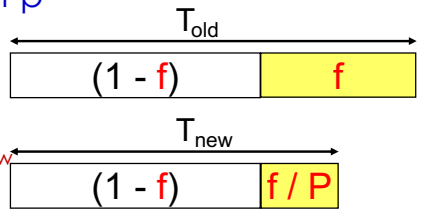
- Sequence 1: IC = 5
  - Clock Cycles = 2×1 + 1×2 + 2×3 = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles = 4×1 + 1×2 + 1×3 = 9
  - Avg. CPI = 9/6 = 1.5

Processor Benchmarking

- Programs used to measure performance
  - Supposedly typical of actual workload
- Examples
  - Standard Performance Evaluation Corp (SPEC)
    - Develops benchmarks for CPU, I/O, Web, …
  - Media Bench - Multimedia
  - EEMBC – Embedded systems
  - Rodinia, Parboil: For GPU Systems
  - SPECWeb, SPECJbb – Enterprise systems
  - Cloudsuite -- Datacenters
  - Many more……

Pitfall: Amdahl's Law

- Suppose $Processor_{old}$ has Runtime of X
  - $Processor_{new}$ can speedup a fraction "f" of the instructions by a factor of p



Speed-up?

$Exec\_time_{old} / Exec\_time_{new}$ = 1/ (1-f + f/p)

Max Speedup?

1/(1-f) *E.g., if f = 90%, max speedup = 10X*

Max speedup of any processor is limited by Amdahl's law

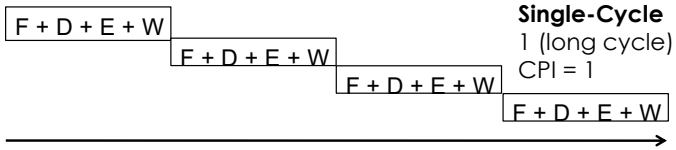
Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

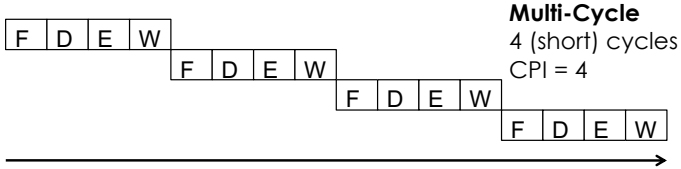
this lecture → ***Sections 4.5-4.10***

Microarchitecture	CPI	cycle time
Single-cycle unpipelined	1	long
Pipelined	1	short
Multi-cycle/Micro-coded	>1	short

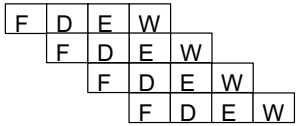
Single-Cycle vs Multi-Cycle vs Pipelined for 4 Independent ADDs



Single-Cycle
1 (long cycle) per instruction
CPI = 1

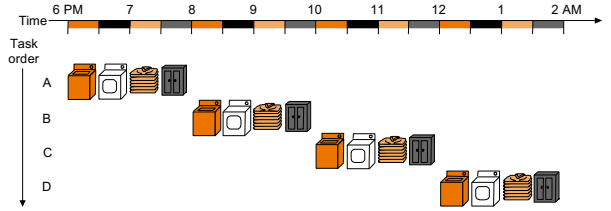


Multi-Cycle
4 (short) cycles per instruction
CPI = 4



Pipelined
4 (short) cycles per 4 instructions
(at steady state)
CPI = 1

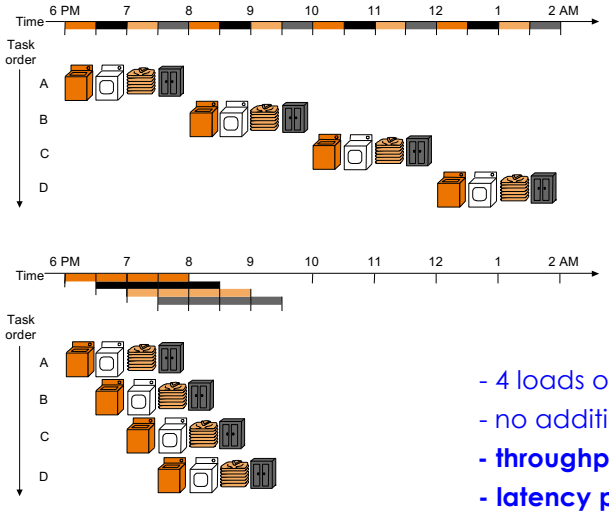
The Laundry Analogy



- "place one dirty load of clothes in the washer"
- "when the washer is finished, place the wet load in the dryer"
- "when the dryer is finished, take out the dry load and fold"
- "when folding is finished, ask your roommate (??) to put the clothes away"

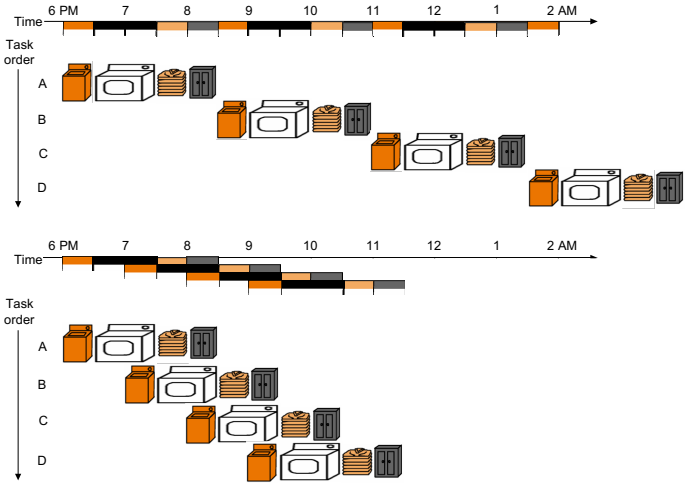
- steps to do a load are sequentially dependent
- no dependence between different loads
- different steps do not share resources

Pipelining Multiple Loads of Laundry



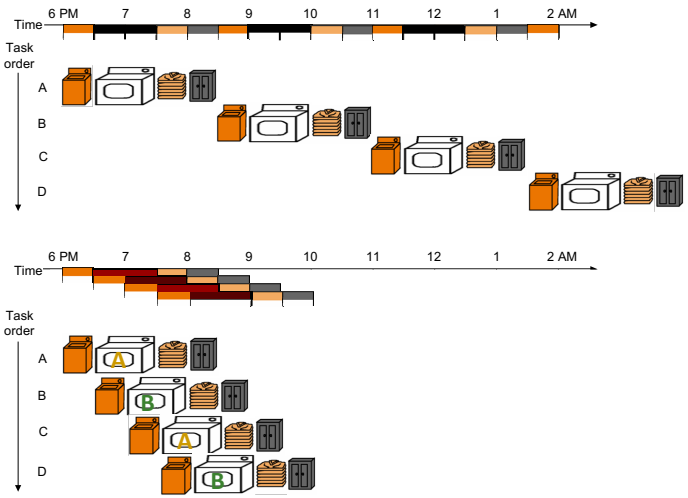
- 4 loads of laundry in parallel
- no additional resources
- **throughput increased by 4**
- **latency per load is the same**

Pipelining Multiple Loads of Laundry: In Practice



the slowest step decides throughput

Pipelining Multiple Loads of Laundry: In Practice

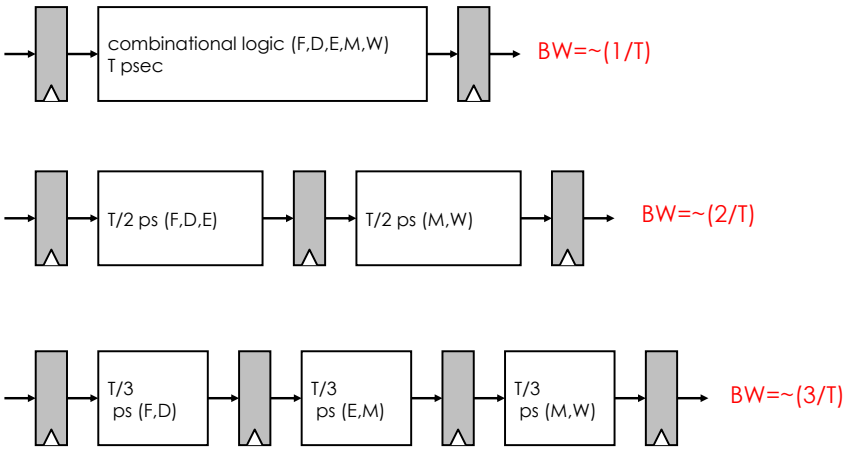


throughput restored (2 loads per hour) using 2 dryers

An "Ideal" Pipeline

- Goal: Increase throughput with little increase in cost (hardware cost, in case of instruction processing)
- Repetition of identical operations
  - The same operation is repeated on a large number of different inputs (e.g., all laundry loads go through the same steps)
- Repetition of independent operations
  - No dependencies between repeated operations
- Uniformly partitionable suboperations
  - Processing can be evenly divided into uniform-latency suboperations (that do not share resources)
- Fitting examples: automobile assembly line, doing laundry
  - What about the instruction processing "cycle"?

Ideal Pipelining



More Realistic Pipeline: Throughput

- Nonpipelined version with delay T

BW = 1/(T+S) where S = latch delay

- k-stage pipelined version

$BW_{k\text{-stage}} = 1 / (T/k + S)$

Latch delay reduces throughput (switching overhead b/w stages)

More Realistic Pipeline: Cost

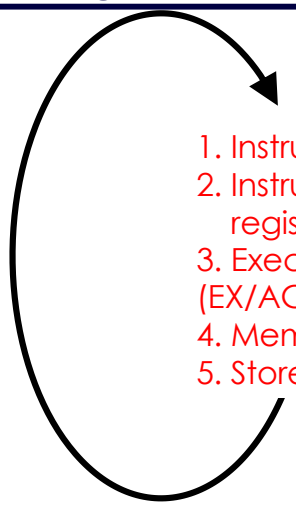
- Nonpipelined version with combinational cost G

  Cost = G+L where L = latch cost

- k-stage pipelined version

  Cost = G + Lk

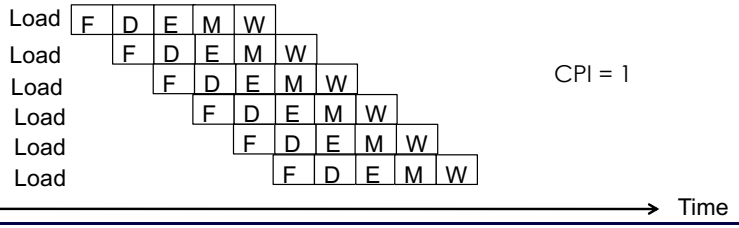
5-Stage Processor Pipeline



1. Instruction fetch (IF)
2. Instruction decode and register operand fetch (ID/RF)
3. Execute/Evaluate memory address (EX/AG)
4. Memory operand fetch (MEM)
5. Store/writeback result (WB)

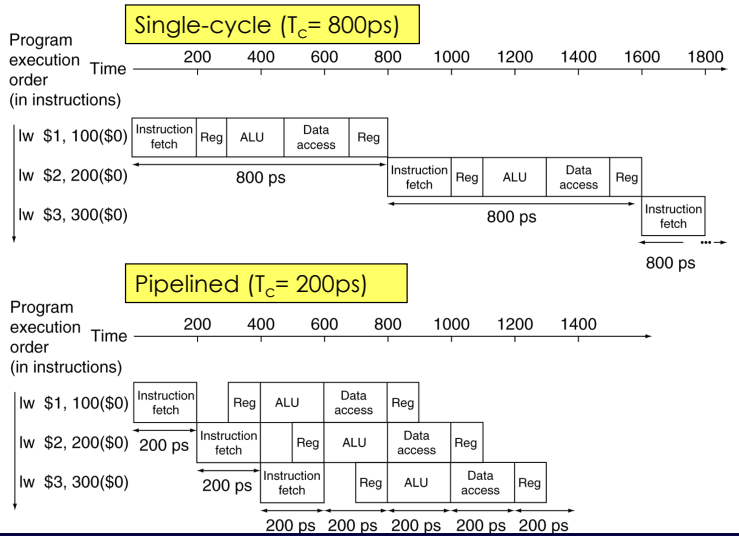
Pipelined Pipeline Performance

Instruction Class	Instruction Fetch 2ns	Decode + Reg 1ns	ALU Operation 2ns	Memory Access 2ns	Register Write 1ns
ALU	✓	✓	✓		✓
Load	✓	✓	✓	✓	✓
Store	✓	✓	✓	✓	
Branch	✓	✓	✓		

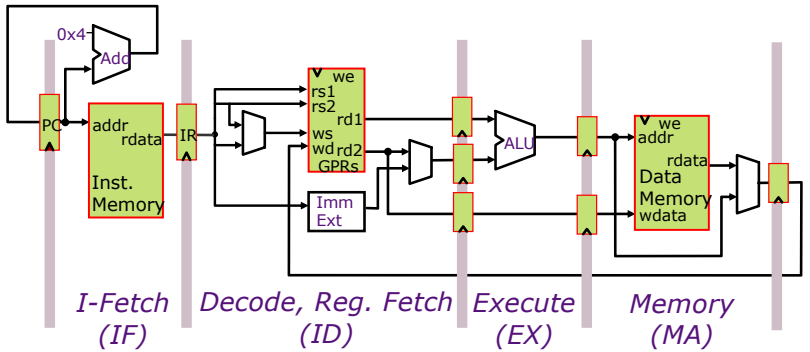


CPI = 1

Pipeline Performance

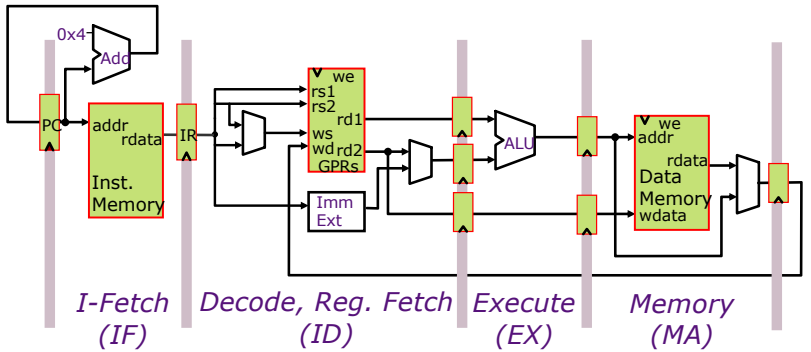


Operation View



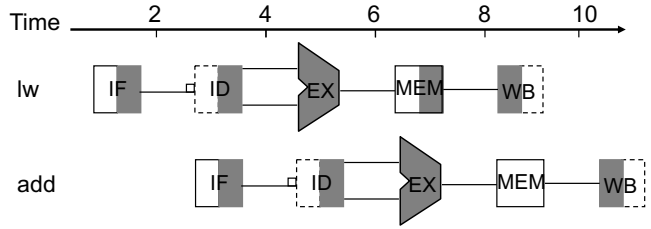
time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
instruction1	IF_1	ID_1	EX_1	MA_1	WB_1				
instruction2		IF_2	ID_2	EX_2	MA_2	WB_2			
instruction3			IF_3	ID_3	EX_3	MA_3	WB_3		
instruction4				IF_4	ID_4	EX_4	MA_4	WB_4	
instruction5					IF_5	ID_5	EX_5	MA_5	WB_5

Resource View



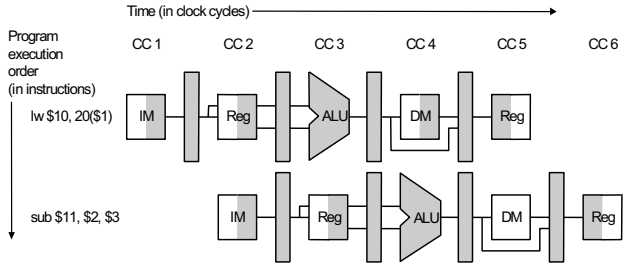
Resources	time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4	I_5				
	ID		I_1	I_2	I_3	I_4	I_5			
	EX			I_1	I_2	I_3	I_4	I_5		
	MA				I_1	I_2	I_3	I_4	I_5	
	WB					I_1	I_2	I_3	I_4	I_5

Graphically Representing Pipelines



- Shading indicates the unit is being used by the instruction
  - Shading on the right half of the register file (ID or WB) or memory means the element is being read in that stage
  - Shading on the left half means the element is being written in that stage

Graphically Representing Pipelines



- Can help with answering questions like:
  - how many cycles does it take to execute this code?
    - 6 cycles
  - what is the ALU doing during cycle 4?
    - Executing subtract operation

Pipelining Example