CPU Performance

COE 233

Logic Design and Computer Organization

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Presentation Outline

- ❖ **Response Time and Throughput**
- ❖ **CPU Performance Equation**
- ❖ Single-Cycle versus Multi-Cycle CPU Performance
- ❖ MIPS Performance Metric
- ❖ Amdahl’s Law

Response Time and Throughput

- Response Time
  - Time between start and completion of a task, as observed by end user
  - Response Time = CPU Time + Waiting Time (I/O, OS scheduling, etc.)
- Throughput
  - Number of tasks the machine can run in a given period of time
- Decreasing execution time improves throughput
  - Example: using a faster version of a processor
  - Less time to run a task ⇒ more tasks can be executed
- Increasing throughput can also improve response time
  - Example: increasing number of processors in a multiprocessor
  - More tasks can be executed in parallel
  - Execution time of individual sequential tasks is not changed
  - But less waiting time in scheduling queue reduces response time

Higher Performance = Less Execution Time

❖ For some program running on machine *X*

$$\boxed{\text{Performance}_X = \frac{1}{\text{Execution time}_X}}$$

❖ *X* is *n* times faster than *Y*

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

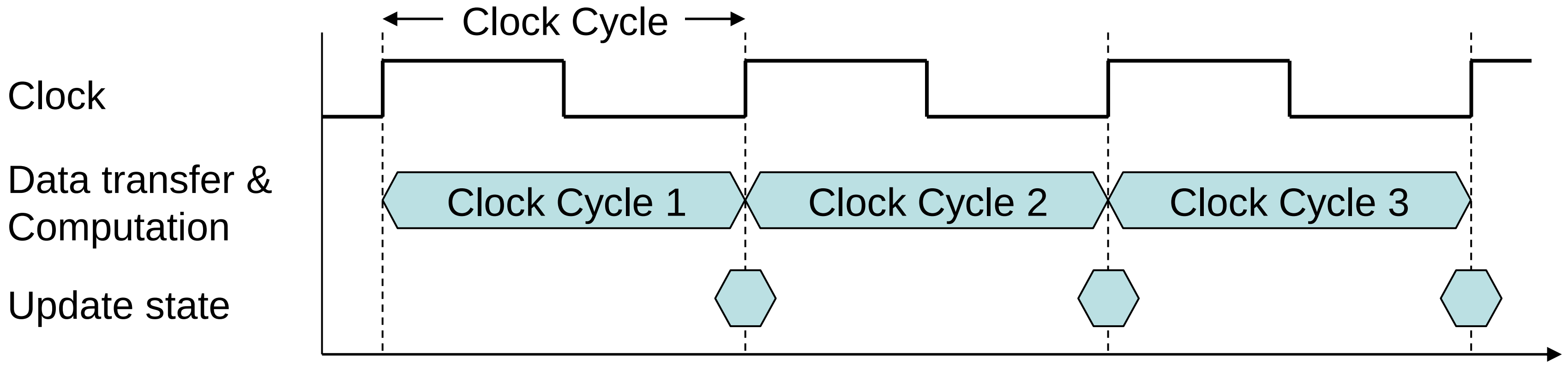
What do we mean by Execution Time?

- Real Elapsed Time
  - Counts everything:
    - Waiting time, Input/output, disk access, OS scheduling, … etc.
  - Useful number, but often not good for comparison purposes
- Our Focus: CPU Execution Time
  - Time spent while executing the program instructions
  - Doesn't count the waiting time for I/O or OS scheduling
  - Can be measured in seconds, or
  - Can be related to number of CPU clock cycles

$$\boxed{\text{CPU Execution Time} = \text{CPU cycles} \times \text{Cycle time} = \frac{\text{CPU cycles}}{\text{Clock rate}}}$$

What is the Clock Cycle?

❖ Operation of digital hardware is governed by a clock



❖ Clock Cycle = Clock period

- ✧ Duration between two consecutive rising edges of the clock signal

❖ Clock rate = Clock frequency = 1 / Clock Cycle

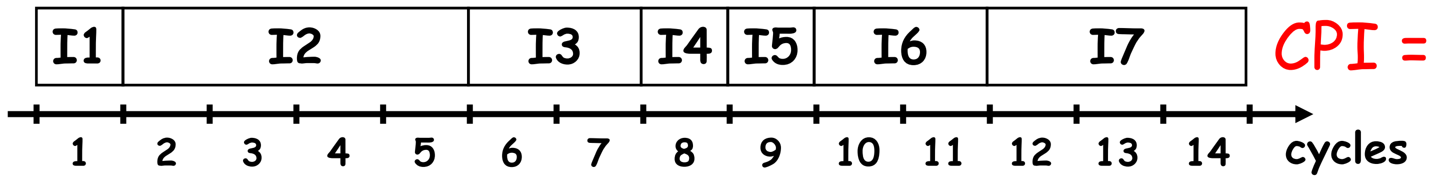
- ✧ 1 Hz = 1 cycle/sec　　1 KHz = 10^3 cycles/sec
- ✧ 1 MHz = 10^6 cycles/sec　　1 GHz = 10^9 cycles/sec
- ✧ 2 GHz clock has a cycle time = $1/(2\times10^9)$ = 0.5 nanosecond (ns)

Improving Performance

- To improve performance, we need to
  - Reduce the number of clock cycles required by a program, or
  - Reduce the clock cycle time (increase the clock rate)
- Example:
  - A program runs in 10 seconds on computer *X* with 2 GHz clock
  - What is the number of CPU cycles on computer *X* ?
  - We want to design computer *Y* to run same program in 6 seconds
  - But computer *Y* requires 10% more cycles to execute program
  - What is the clock rate for computer *Y* ?
- Solution:
  - CPU cycles on computer *X* = 10 sec × 2 × 10^9 cycles/s = 20 × 10^9 cycles
  - CPU cycles on computer *Y* = 1.1 × 20 × 10^9 = 22 × 10^9 cycles
  - Clock rate for computer *Y* = 22 × 10^9 cycles / 6 sec = 3.67 GHz

Clock Cycles per Instruction (CPI)

- Instructions take different number of cycles to execute
  - Multiplication takes more time than addition
  - Floating point operations take longer than integer ones
  - Accessing memory takes more time than accessing registers
- CPI is an average number of clock cycles per instruction



CPI = 14/7 = 2.0

- Important point

Changing the cycle time often changes the number of cycles required for various instructions

Performance Equation

- To execute, a given program will require …
  - Some number of machine instructions
  - Some number of clock cycles
  - Some number of seconds
- We can relate CPU clock cycles to instruction count

$$\boxed{\text{CPU cycles} = \text{Instruction Count} \times \text{CPI}}$$

- Performance Equation: (related to instruction count)

$$\boxed{\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Cycle time}}$$

Understanding Performance Equation

Execution Time = Instruction Count × CPI × Cycle time

	I-Count	CPI	Cycle
Program	X		
Compiler	X	X	
ISA	X	X	
Organization		X	X
Technology			X

Using the Performance Equation

- Suppose we have two implementations of the same ISA
- For a given program
  - Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  - Machine B has a clock cycle time of 500 ps and a CPI of 1.2
  - Which machine is faster for this program, and by how much?
- Solution:
  - Both computer execute same count of instructions = I
  - CPU execution time (A) = I × 2.0 × 250 ps = 500 × I ps
  - CPU execution time (B) = I × 1.2 × 500 ps = 600 × I ps
  - Computer A is faster than B by a factor = $\dfrac{600 \times I}{500 \times I} = 1.2$

Determining the CPI

- ❖ Different types of instructions have different CPI

  Let CPI_i = clocks per instruction for class *i* of instructions

  Let C_i = instruction count for class *i* of instructions

$$\boxed{\text{CPU cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)}$$

$$\boxed{CPI = \frac{\sum_{i=1}^{n} (CPI_i \times C_i)}{\sum_{i=1}^{n} C_i}}$$

- ❖ Designers often obtain CPI by a detailed simulation
- ❖ Hardware counters are also used for operational CPUs

Example on Determining the CPI

- ❖ Problem

A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: class A, class B, and class C, and they require one, two, and three cycles per instruction, respectively.

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C

The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C

Compute the CPU cycles for each sequence. Which sequence is faster?

What is the CPI for each sequence?

- ❖ Solution

CPU cycles (1st sequence) = $(2\times1) + (1\times2) + (2\times3) = 2+2+6 = 10$ cycles

CPU cycles (2nd sequence) = $(4\times1) + (1\times2) + (1\times3) = 4+2+3 = 9$ cycles

Second sequence is faster, even though it executes one extra instruction

CPI (1st sequence) = $10/5 = 2$ CPI (2nd sequence) = $9/6 = 1.5$

Second Example on CPI

Given: instruction mix of a program on a RISC processor

What is average CPI?

What is the percent of time used by each instruction class?

Class_i	Freq_i	CPI_i	$\text{CPI}_i \times \text{Freq}_i$	%Time
ALU	50%	1	0.5×1 = 0.5	0.5/2.2 = 23%
Load	20%	5	0.2×5 = 1.0	1.0/2.2 = 45%
Store	10%	3	0.1×3 = 0.3	0.3/2.2 = 14%
Branch	20%	2	0.2×2 = 0.4	0.4/2.2 = 18%

Average CPI = 0.5+1.0+0.3+0.4 = 2.2

How faster would the machine be if load time is 2 cycles?

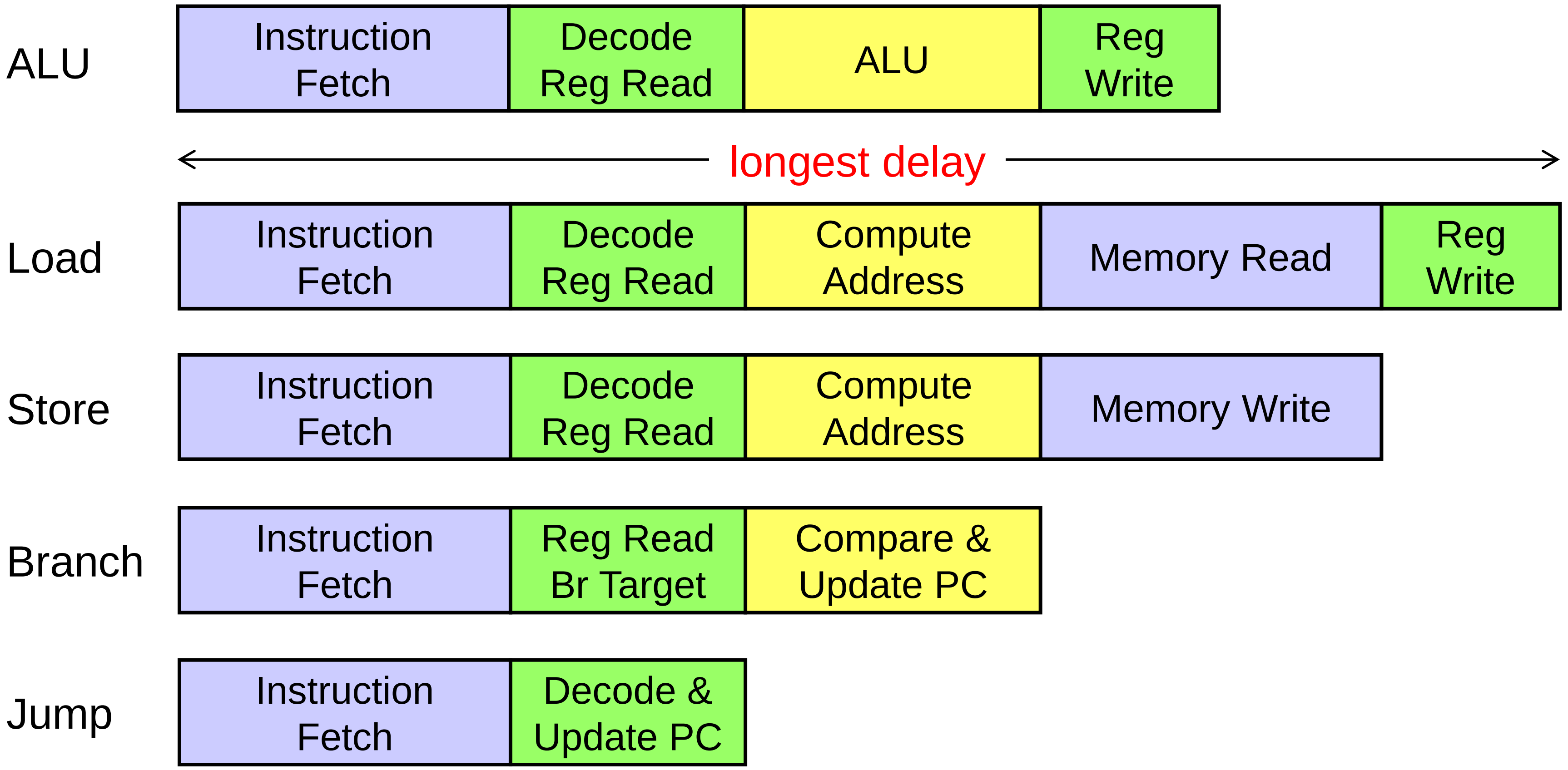
What if two ALU instructions could be executed at once?

Presentation Outline

- Response Time and Throughput
- CPU Performance Equation
- **Single-Cycle versus Multi-Cycle CPU Performance**
- **MIPS Performance Metric**
- **Amdahl’s Law**

Drawback of Single Cycle Processor

- Single Cycle ➔ CPI = 1 for all instructions
- Major drawback is the **Long cycle time**
- All instructions take as much time as the **slowest instruction**



Alternative: Multicycle Implementation

- Break instruction execution into multiple stages
  - Instruction fetch
  - Instruction decode, register read, target address for jump/branch
  - Execution, memory address calculation, or branch outcome
  - Memory access or ALU instruction completion
  - Load instruction completion
- One clock cycle per stage (clock cycle is reduced)
  - First 2 stages are the same for all instructions

Instruction	# cycles	Instruction	# cycles
ALU & Store	4	Branch	3
Load	5	Jump	2

Single-Cycle versus Multi-Cycle Performance

- Assume the following operation times for components:
  - Access time for Instruction and data memories: 200 ps
  - Delay in ALU and adders: 180 ps
  - Delay in Decode and Register file access (read or write): 150 ps
  - Ignore the other delays in PC, mux, extender, and wires
- Which of the following would be faster and by how much?
  - Single-cycle implementation for all instructions
  - Multi-cycle implementation optimized for every class of instructions
    - Load = 5 cycles, ALU = Store = 4 cycles, Branch = 3 cycles, Jump = 2 cycles
- Assume the following instruction mix:
  - 40% ALU, 20% Loads, 10% stores, 20% branches, & 10% jumps

Solution

Instruction Class	Instruction Memory	Register Read	ALU Operation	Data Memory	Register Write	Total
ALU	200	150	180		150	680 ps
Load	200	150	180	200	150	880 ps
Store	200	150	180	200		730 ps
Branch	200	150	180 ←	Compare and update PC		530 ps
Jump	200	150 ←	Decode and update PC			350 ps

- For fixed single-cycle implementation:
  - Clock cycle = 880 ps determined by longest delay (load instruction)
- For multi-cycle implementation:
  - Clock cycle = max (200, 150, 180) = 200 ps (maximum delay at any step)
  - Average CPI = $0.4 \times 4 + 0.2 \times 5 + 0.1 \times 4 + 0.2 \times 3 + 0.1 \times 2 = 3.8$
- Speedup = $(1 \times 880 \text{ ps}) / (3.8 \times 200 \text{ ps}) = 880 / 760 = 1.16$

MIPS Performance Metric

- MIPS: Millions Instructions Per Second
- Sometimes used as performance metric
  - Faster machine $\Rightarrow$ larger MIPS
- MIPS specifies instruction execution rate

$$\boxed{\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}}$$

- We can also relate execution time to MIPS

$$\boxed{\text{Execution Time} = \frac{\text{Inst Count}}{\text{MIPS} \times 10^6} = \frac{\text{Inst Count} \times \text{CPI}}{\text{Clock Rate}}}$$

Drawbacks of MIPS

Three problems using MIPS as a performance metric

1. Does not take into account the capability of instructions
   - ✧ Cannot use MIPS to compare computers with different instruction sets because the instruction count will differ
2. MIPS varies between programs on the same computer
   - ✧ A computer cannot have a single MIPS rating for all programs
3. MIPS can vary inversely with performance
   - ✧ A higher MIPS rating does not always mean better performance
   - ✧ Example in next slide shows this anomalous behavior

MIPS example

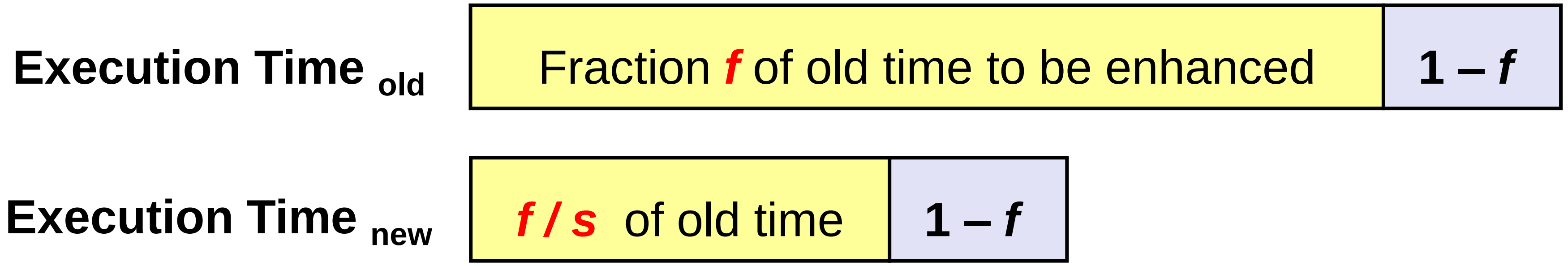
- ❖ Two different compilers are being tested on the same program for a 4 GHz machine with three different classes of instructions: Class A, Class B, and Class C, which require 1, 2, and 3 cycles, respectively.
- ❖ The instruction count produced by the first compiler is 5 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- ❖ The second compiler produces 10 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- ❖ Which compiler produces a higher MIPS?
- ❖ Which compiler produces a better execution time?

Solution to MIPS Example

- First, we find the CPU cycles for both compilers
  - CPU cycles (compiler 1) = $(5\times1 + 1\times2 + 1\times3)\times10^9 = 10\times10^9$
  - CPU cycles (compiler 2) = $(10\times1 + 1\times2 + 1\times3)\times10^9 = 15\times10^9$
- Next, we find the execution time for both compilers
  - Execution time (compiler 1) = 10×10^9 cycles / 4×10^9 Hz = 2.5 sec
  - Execution time (compiler 2) = 15×10^9 cycles / 4×10^9 Hz = 3.75 sec
- Compiler1 generates faster program (less execution time)
- Now, we compute MIPS rate for both compilers
  - MIPS = Instruction Count / (Execution Time $\times 10^6$)
  - MIPS (compiler 1) = $(5+1+1) \times 10^9 / (2.5 \times 10^6) = 2800$
  - MIPS (compiler 2) = $(10+1+1) \times 10^9 / (3.75 \times 10^6) = 3200$
- So, code from compiler 2 has a higher MIPS rating !!!

Amdahl's Law

- Amdahl's Law is a measure of Speedup
  - How a program performs after improving portion of a computer
  - Relative to how it performed previously
- Let ***f*** = Fraction of the computation time that is enhanced
- Let ***s*** = Speedup factor of the enhancement only



$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{((1 - f) + f / s)}$$

Example on Amdahl's Law

- Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?
- Solution: suppose we improve multiplication by a factor s

  25 sec (4 times faster) = 80 sec / s + 20 sec

  s = 80 / (25 – 20) = 80 / 5 = 16

  Improve the speed of multiplication by s = 16 times
- How about making the program 5 times faster?

  20 sec (5 times faster) = 80 sec / s + 20 sec

  $s = 80 / (20 – 20) = \infty$ Impossible to make 5 times faster!

Example 2 on Amdahl's Law

- Suppose that floating-point square root is responsible for 20% of the execution time of a graphics benchmark and ALL FP instructions are responsible for 60%
- One proposal is to speedup FP SQRT by a factor of 10
- Alternative choice: make ALL FP instructions 2X faster, which choice is better?
- Answer:
  - Choice 1: Improve FP SQRT by a factor of 10
  - Speedup (FP SQRT) = 1/(0.8 + 0.2/10) = 1.22
  - Choice 2: Improve ALL FP instructions by a factor of 2
  - Speedup = 1/(0.4 + 0.6/2) = 1.43 ➔ **Better**