Computer Architecture

3. Performance

Young Geun Kim
(younggeun_kim@korea.ac.kr)
Intelligent Computer Architecture & Systems Lab.

How to Define Performance?

- There are many ways to define something as “the best”
- Airplane Example

Airplane	Passenger Capacity	Cruising Range (miles)	Cruising Speed (m.p.h)	Throughput (passengers * m.p.h)
Boeing 777	375	5,256	610	228,750
Boeing 747	416	7,156	610	286,700
Airbus 380	525	8,200	560	294,000
BAC/Sud Concorde	132	4,000	1,350	178,200
Douglas DC-8-50	146	8,720	544	79,424

- Which is “the best”?

Applying to Computers

- **How do you decide which computer is the best?**
  - Processor Speed
  - Memory Size
  - Storage Speed
  - Graphic Card / Subsystem
  - Energy Efficiency
  - Price
  - Weight

- **How do you decide which to buy?**
  - Trade-off between cost and performance
    - Recently, energy has been added (esp. for battery-powered systems).

Example of Metrics in Computers

Application program	Answers a month Operations per second
Compiler	(Millions) of instructions per second: MIPS
Operating System	(Giga) (F.P.) operations per second: GFLOPS/s
ISA	
Data path / control	Cycles per instruction: CPI
Functional units	
Logic / transistors	Clock frequency: GHz

⇒ 시간과 관련

- **Most of metrics are related to time (how fast)**
  - Time is the most classical metric for computers

Measuring Time

- **Looking at measuring CPU performance, we are primarily concerned with execution time.**

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

- **To compare, we say "X is n times faster than Y"**

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

- **Increased performance = Reduced execution time**
  - Improved performance = Improved execution time

Example of Performance Comparison

- **X and Y do their homework**
  - X takes 5 hours
  - Y takes 10 hours

- **Compare the performance**

$$\text{Performance}_X = \frac{1}{\text{Execution Time}_X} = \frac{1}{5\text{ hours}} = 0.2$$

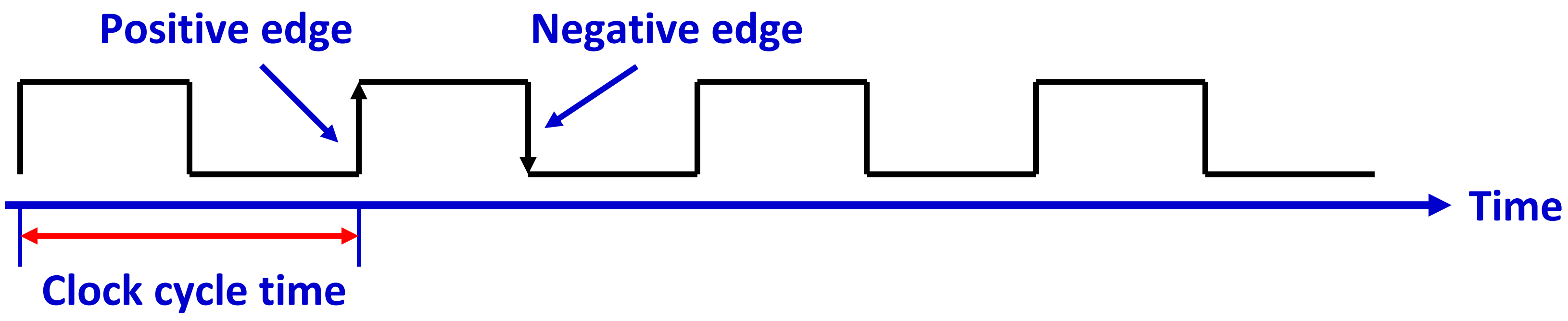
$$\text{Performance}_Y = \frac{1}{\text{Execution Time}_Y} = \frac{1}{10\text{ hours}} = 0.1$$

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{0.2}{0.1} = 2$$

- **So, X is two times faster than Y**

Clock Cycle Time vs. Clock Rate

- **Clock Cycle Time** ⇒ 프로세서에 따라 정해져 있음.
  - Time required for a clock pulse to make transitions: 0 -> 1 -> 0

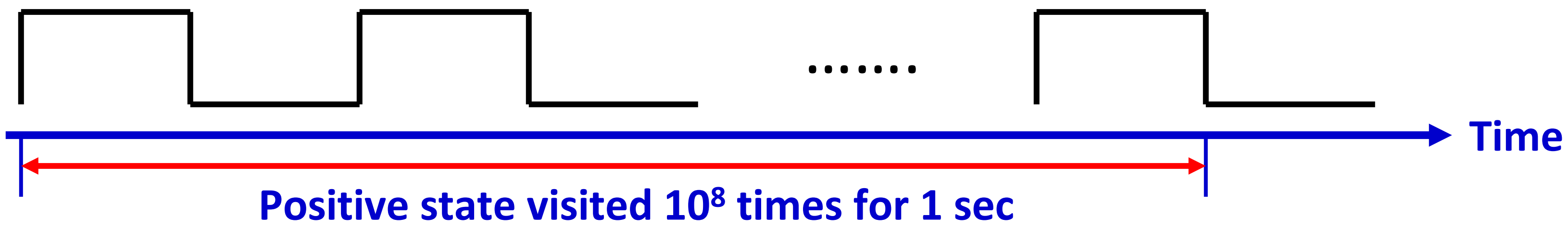


  - In other words, the time duration between positive (negative) edges

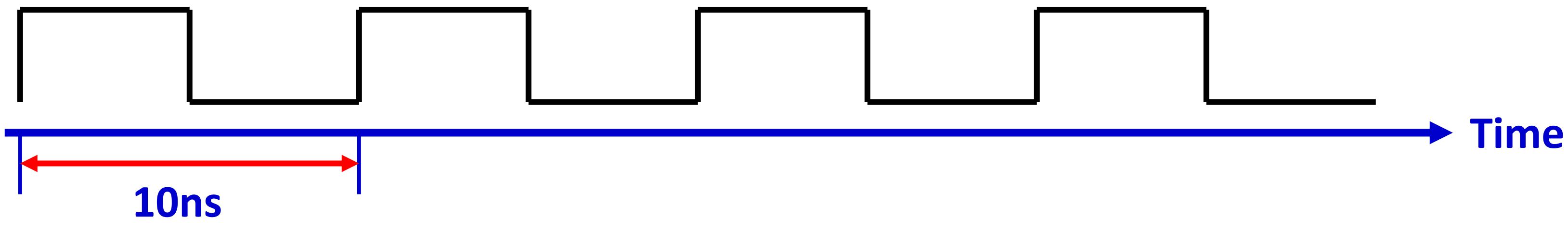
- **Clock Rate**
  - Inverse of clock cycle time (= 1 / clock cycle time)
  - # of times to visit positive (negative) state per second
  - Unit: Hz or MHz or GHz

Example of Clock Cycle Time

- **A Machine is running at 100MHz**
  - Clock rate = 100 MHz = 100 * 10^6 cycles / sec

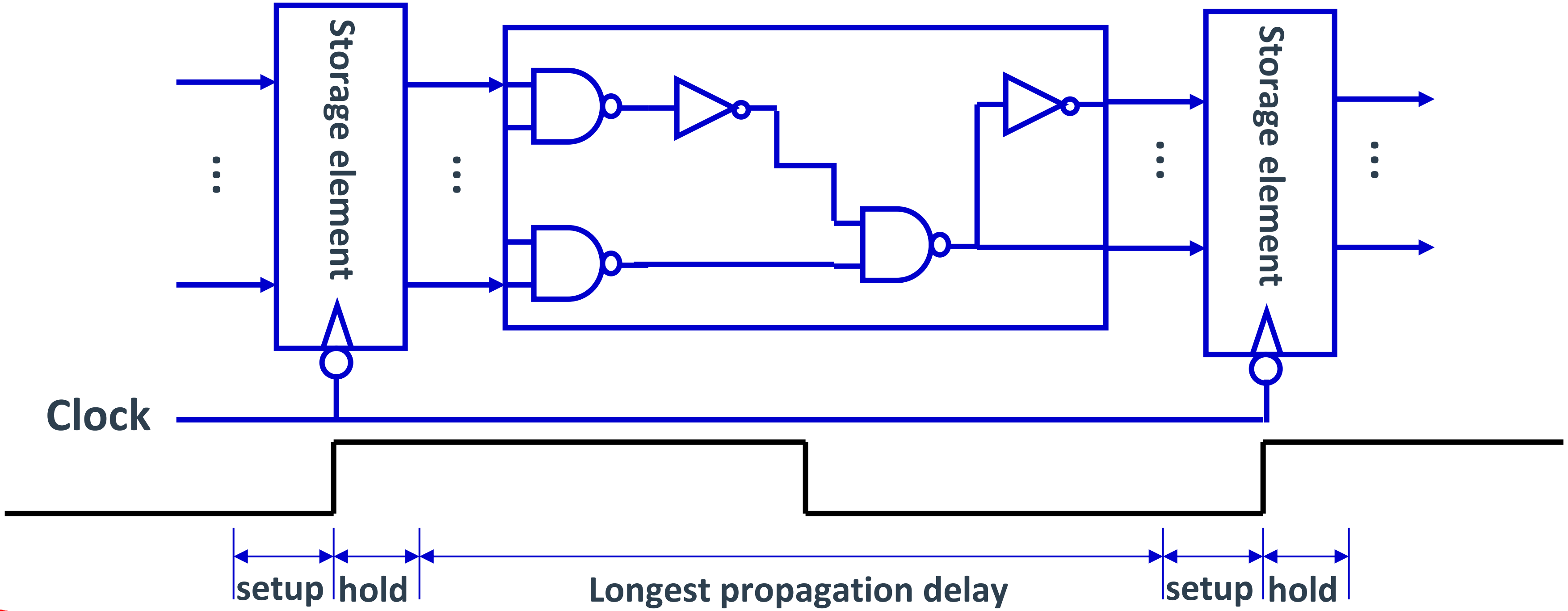


  - Clock cycle time = 1 / {(100 * 10^6) cycles / sec} = 10 ns



How is clock cycle time determined?

- **Closely related to logic design**



- **Longest propagation delay** ⇒ 모든 gate를 통과하는 데 걸리는 시간 (최장 시간)
  - a.k.a. critical path delay
  - Critical path: a path that takes the longest timing delays among many combinational paths

Execution Time

- **We will use CPU execution time frequently as the metric of how long a program should run.**

Execution time = Clock cycles for program * Clock cycle time

- **Since clock cycle time is the inverse of clock rate**

$$\text{Execution time} = \frac{\text{Clock cycles for program}}{\text{Clock rate}}$$

Measuring clock cycles

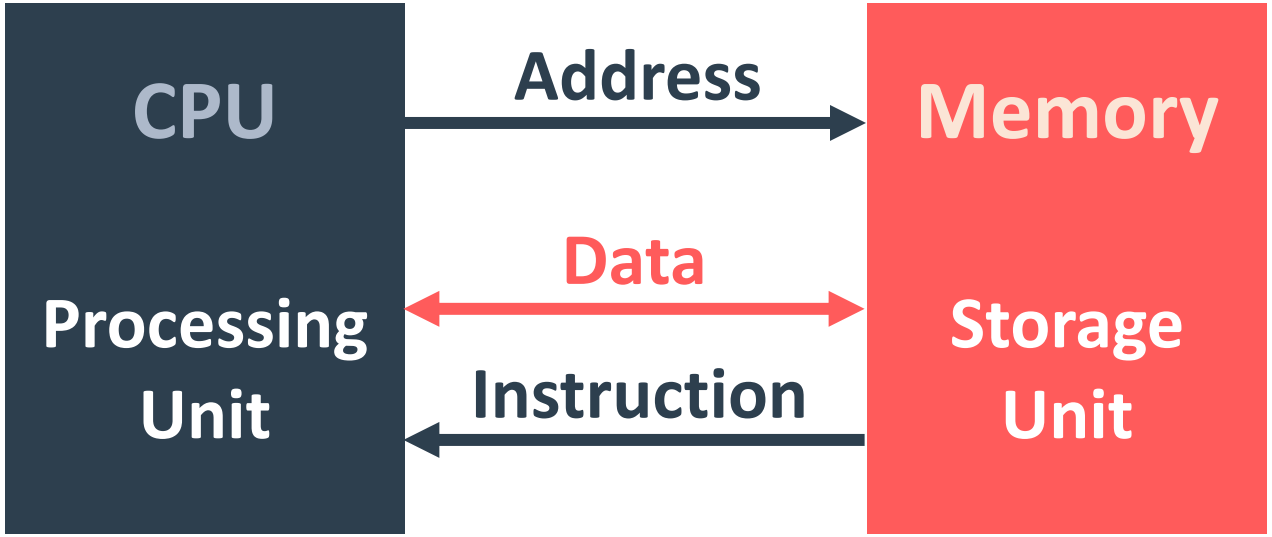
- **CPU clock cycles / program is not so intuitive.**
  - It is very difficult to estimate # of CPU clock cycles for your program.

Unit of basic operations (addition, subtraction...)

- **CPI (Cycles Per Instruction) is used so frequently.**
  - The # of cycles per instruction varies depending on the instructions, so CPI is an average value.
  - CPIs can be compared between ISAs
  - 프로세서에 따라 정해져 있음.

Von Neumann Architecture

- By John von Neumann, 1945



③ ① Data Movement

② Arithmetic & Logical Ops

Control Transfer (e.g., if, while, for, …)

Byte Addressable Array

Code + Data

Stack to Support Procedures

Using CPI

- **Therefore, we can rewrite**
  - $\boxed{\text{Execution time} = \#\text{ of Instructions} * \text{CPI} * \text{Clock cycle time}}$

$$\boxed{\text{Execution time} = \frac{\text{Clock cycles for program}}{\text{Clock rate}}}$$

  - **Improved performance (= reduced execution time) is possible** with **increased clock rate** (= reduced clock cycle time), **lower CPI**, or **reduced # of instruction**.

직접 바꿀 수 없는 값

  - Designers have to balance the length of each cycle and # of cycles required.

CPI Examples

- Machine A: 1ns clock and CPI of 2.0
- Machine B: 2ns clock and CPI of 1.2
- Which is faster?

Example Solution

- **Solve CPU time for each machine**
  - Execution $time_A$ = I (= # of Instructions) * 2.0 * 1 ns = 2.0 * I ns
  - Execution $time_B$ = I (= # of Instructions) * 1.2 * 2 ns = 2.4 * I ns

- **Compare performance**

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{2.4 * \text{I ns}}{2.0 * \text{I ns}} = 1.2$$

- **So, machine A is 1.2 times faster than machine B**
  - CPI is not always correct…
  - Why?

CPI Variability

- Different types of instructions often take different numbers of cycles on the same processor.
- CPI is often reported for classes of instructions
  - $$\boxed{CPI = \frac{\sum_{i=1}^{n} (CPI_i \times C_i)}{\text{Total instruction count}}}$$ 평균값
  - CPI_i : the CPI for *i* class of instructions
  - C_i : the count of *i* type of instructions

CPI from Instruction Mix

- CPI = $\dfrac{\sum_{i=1}^{n} (CPI_i \times C_i)}{\text{Total instruction count}}$

- CPI Example

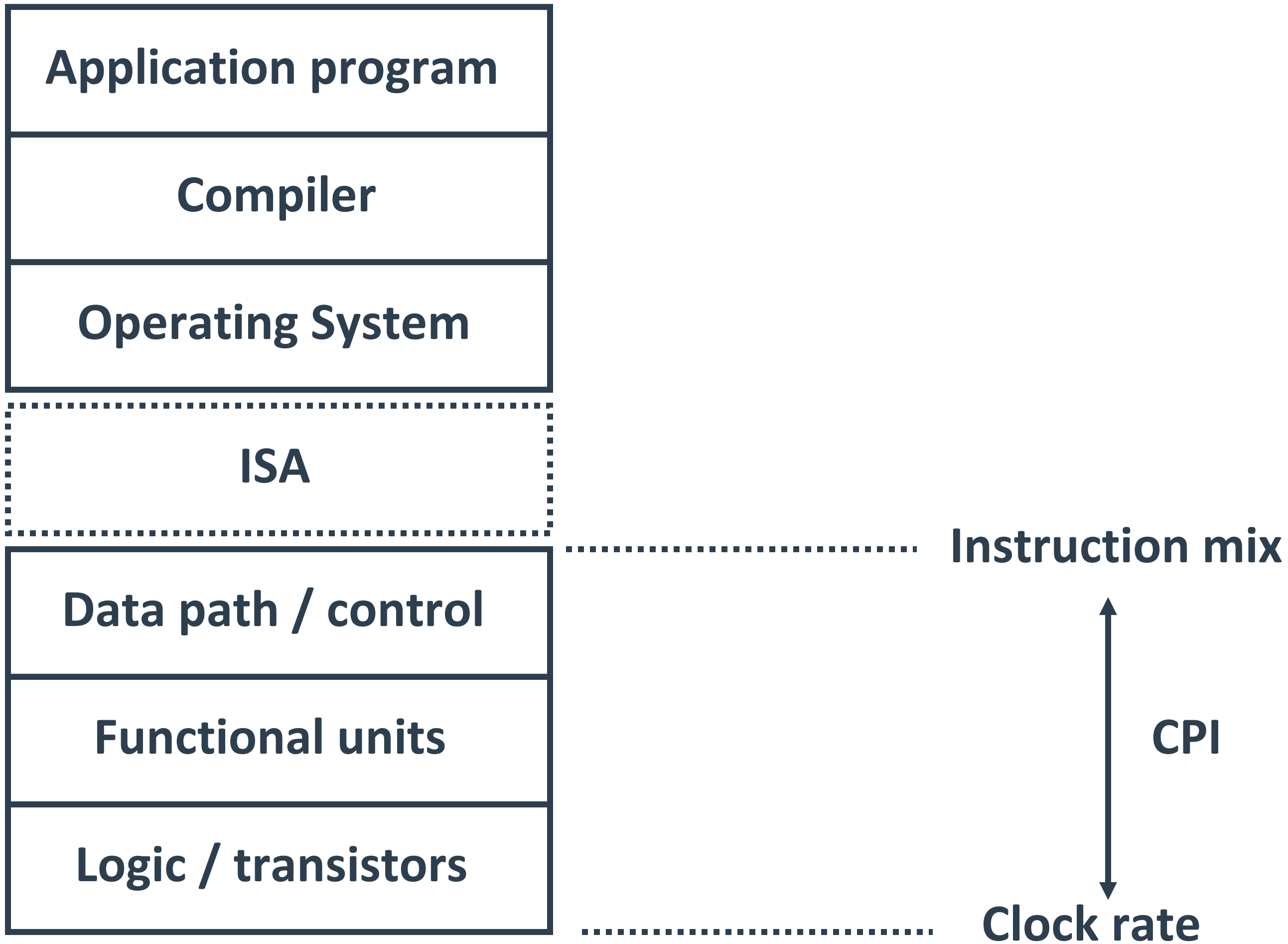
Instruction Class	Frequency	CPI_i
ALU operations	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

CPI = 0.43 x 1 + 0.21 x 2 + 0.12 x 2 + 0.24 x 2 = 1.57

Clock cycles = 1.57 * Instruction Count

Trade-offs

- Instruction count, CPI, and clock rate present trade-offs



Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Instruction count	CPI	Clock rate
Program	V		
Compiler	V		
ISA	V	V	
Organization		V	V
Technology			V

Not just about the program algorithms

Another Popular Performance Metrics

- **MIPS (million instructions per second)**
  - **MIPS** $= \dfrac{\textbf{Instruction Count}}{\textbf{Execution Time x } \mathbf{10^6}}$
  - **Problems**
    - It does not take into account the instruction set.
- **GFLOPS (giga floating-point operations per second)**
- **TFLOPS (tera floating-point operations per second)**
  - Operations rather than instruction
  - e.g., floating-point addition, multiplication, …

A Wrong Use Case of MIPS

- Consider a 500 MHz Machine

Class	CPI
Class A	1
Class B	2
Class C	3

- Consider the two compilers

Code from	Instruction Counts (millions)		
	A	B	C
Compiler1	5	1	1
Compiler2	10	1	1

- Which compiler produce faster code? Has a higher MIPS?

A Wrong Use Case of MIPS: Solution (I)

- **Compute Clock Cycles**
  - Clock cycles = $\sum_{i=1}^{n} (\mathbf{CPI_i \times C_i})$
  - Clock $\text{cycles}_{\text{comp1}}$ = (1 x 5M) + (2 x 1M) + (3 x 1M) = 10M
  - Clock $\text{cycles}_{\text{comp2}}$ = (1 x 10M) + (2 x 1M) + (3 x 1M) = 15M
- **Execution Time**
  - Execution time = (Instruction count x CPI) / Clock rate
    = Clock cycles / Clock rate
  - $\text{Execution}_{\text{comp1}}$ = 10M / 500M = 0.02 s
  - $\text{Execution}_{\text{comp2}}$ = 15M / 500M = 0.03 s
- **Code from compiler 1 is 1.5x faster!**

A Wrong Use Case of MIPS: Solution (II)

- **Computer MIPS**
  - **MIPS** $= \dfrac{\textbf{Instruction Count}}{\textbf{Execution Time x } \mathbf{10^6}}$

  - $\text{MIPS}_{comp1} = (5M + 1M + 1M) / (0.02M) = 350$
  - $\text{MIPS}_{comp2} = (10M + 1M + 1M) / (0.03M) = 400$

- **Code from compile 2 is faster??**
  - Fails to give a right answer!

Benchmarks

- **Users often want a performance metric.**
- **A benchmark is distillation of the attributes of a workload.**
  - Real applications usually work best, but using them is not always feasible.

- **Desirable attributes**
  - Relevant: meaningful within the target domain
  - Understandable
  - Good metric(s)
  - Scalable
  - Coverage: does not oversimplify important factors in the target domain
  - Acceptance: vendors and users embrace it

Benchmarks (cont'd)

- **De-facto industry standard benchmarks for CPU**
  – SPEC

- **SPEC**
  – Standard Performance Evaluation Cooperative
  – Founded in 1988 by EE times, SUN, HP, MIPS, Apollo, DEC
  – Several different SPEC benchmarks
  – Most include a suit of several different applications (such as integer and floating point components often reported separately)
  – For more information, visit http://www.spec.org

Amdahl's Law 부분의 향상은 그 부분이 전체에서 차지하는 비중에 의해 제한된다.

Execution speedup is proportional to the size of the improvement and the amount affected

- Execution time after improvement

$$= \left[\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected} \right]$$

- Or

$$ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

- Hardware-independent metrics do not work (e.g., code size).

Example – Amdahl's Law

- **Floating point instructions improved to run 2x; but only 10% of actual instructions are FP**

unaffected

$$ExTime_{new} = ExTime_{old} \times (0.9 + 0.1/2) = 0.95 \times ExTime_{old}$$

$$Speedup_{overall} = \frac{ExTime_{old}}{0.95 \times ExTime_{old}} = 1.053$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}}$$

Example – Amdahl's Law

- CPI = $$\frac{\sum_{i=1}^{n} (CPI_i \times C_i)}{\text{Total instruction count}}$$

- CPI Example

Instruction Class	Frequency	CPI_i
ALU operations	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

CPI = 0.43 x 1 + 0.21 x 2 + 0.12 x 2 + 0.24 x 2 = 1.57

Clock cycles = 1.57 * Instruction Count