숭실대학교
Soongsil University

Computer Architecture

강의 #2: Performance

2021년 1학기

Young Geun Kim (김영근)

성능의 정의

- "가장 좋음"을 정의하는 방식은 여러가지가 있음
- 비행기 예시

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

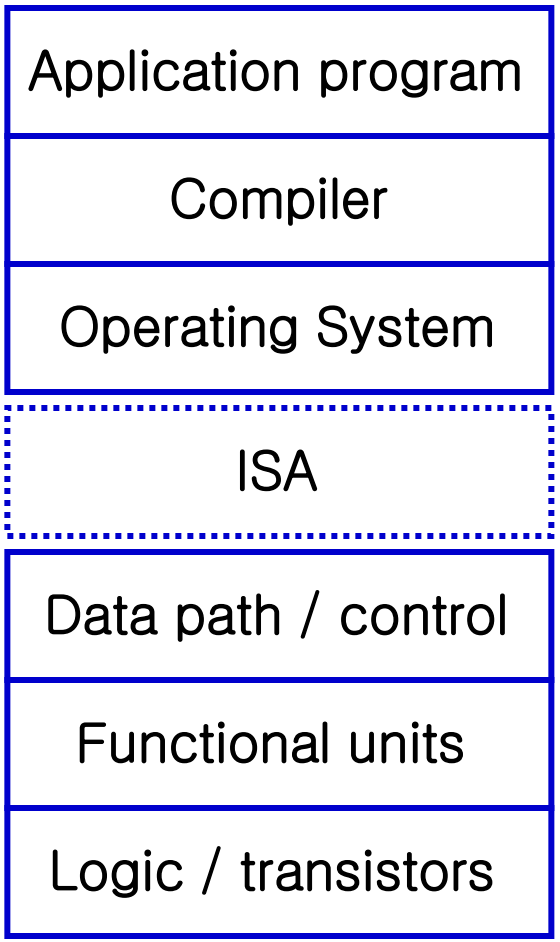
- 어떤 것이 "가장 좋은 것"일까?

컴퓨터에 적용해 보면..

- **언제 컴퓨터가 가장 좋다고 말할 수 있을까?**
  - 프로세서 성능
  - 메모리 크기
  - SSD나 HDD의 성능
  - 그래픽
  - 전력 소모
  - 가격
  - 무게

- **어떤 기준으로 컴퓨터를 구매하는가?**
  - 성능과 비용 사이의 Tradeoff를 고려

컴퓨터에서 성능을 평가하는 기준의 예



- **대부분의 성능 평가 기준은 시간과 관련이 있음 (얼마나 빠른지)**
  - 시간은 컴퓨터의 성능을 평가하기 위한 가장 고전적인 방법임

시간 측정 방법

- CPU의 성능을 측정할 때, 우리는 실행 시간을 주로 고려함

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

- 이를 비교할 때 우리는 "X가 Y보다 n배 빠르다" 라고 함

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

- 성능이 증가하면, 실행 시간은 짧아짐 (= 같은 것을 수행하는데 시간이 덜 걸림)
  - 둘 모두에 대해 우리는 "향상된다"고 말함

성능 비교의 예

- **X와 Y가 과제를 한다고 가정**
  - X는 과제하는데 5시간이 걸림
  - Y는 과제하는데 10시간이 걸림

- **둘 사이의 성능을 비교해보자**

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

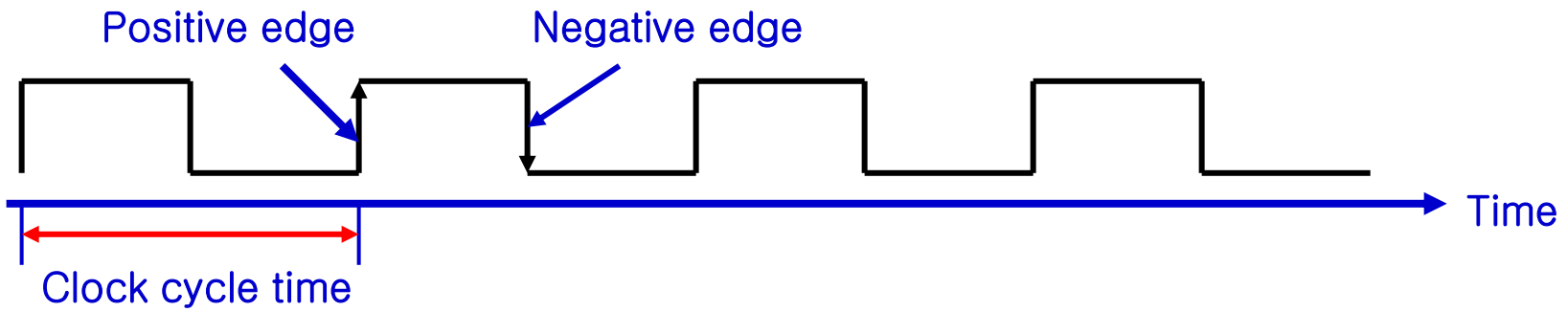
$$\text{Performance}_Y = \frac{1}{\text{Execution Time}_Y} = \frac{1}{10 \text{ hours}} = 0.1$$

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

- **따라서, X가 Y보다 2배 빠르다고 말할 수 있음**

Clock Cycle Time vs. Clock Rate

- **Clock Cycle Time**
  - Clock 신호가 0 -> 1 -> 0 (또는 1 -> 0 -> 1) 으로 변하는데 걸리는 시간

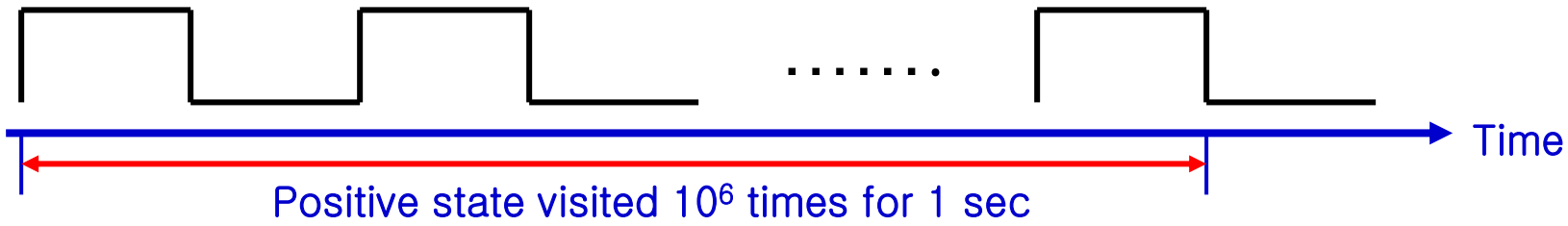


  - 다시 말해, Positive (또는 Negative) Edge 사이의 시간이라고 할 수 있음

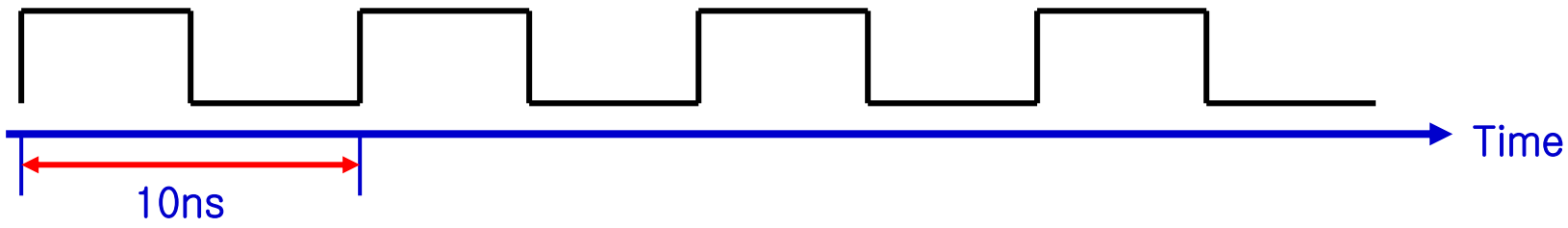
- **Clock Rate**
  - Clock Cycle Time의 역수
  - 1초 동안 Clock 신호가 몇 번 바뀌는지
  - 단위: Hz or MHz or GHz

Clock Cycle Time의 예

- 어떤 기계가 100 MHz로 동작하고 있다면,
  - Clock rate = 100MHz = 100 * 10^6 cycles / sec

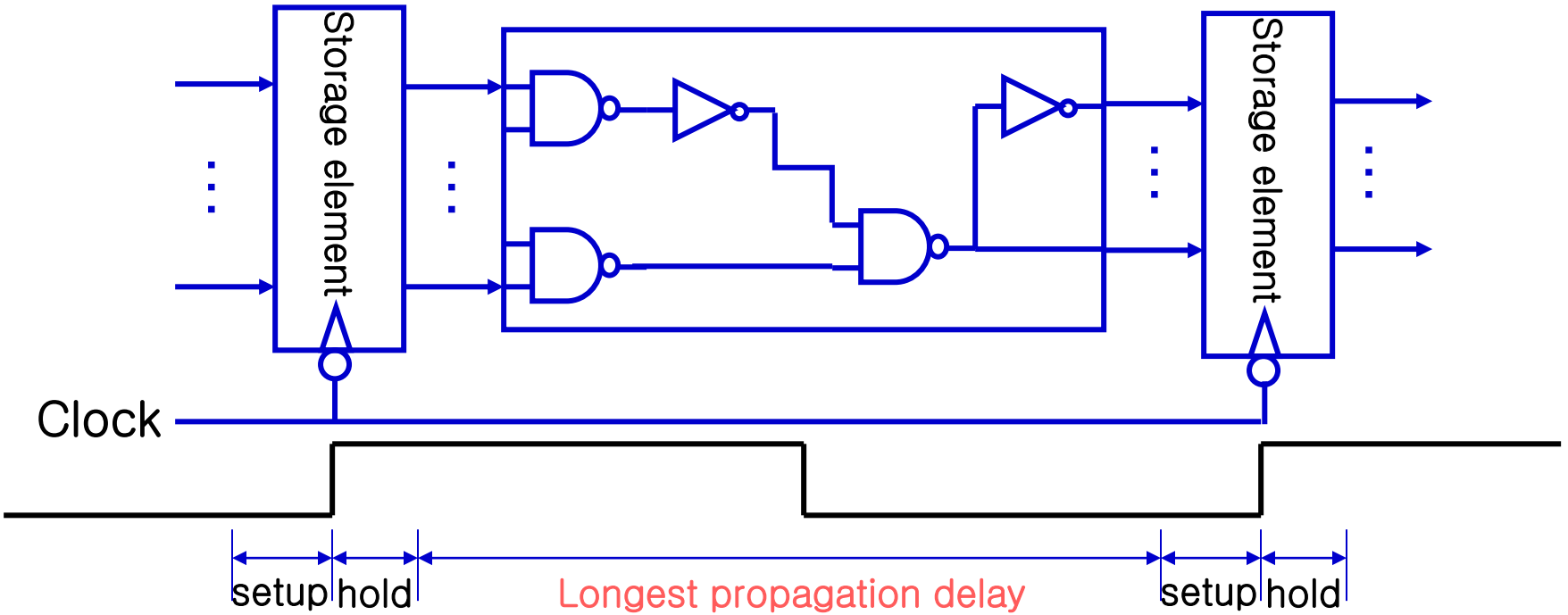


  - Clock cycle time = 1/(100*10^6) cycles / sec = 10ns



Clock Cycle Time은 어떻게 정해지는가?

- 프로세서의 회로와 깊은 관련이 있음
- 가정
  - 프로세서 내의 모든 저장장치는 같은 Clock을 사용함
  - (앞서 나온 것과 같이) Edge에서 Clock이 변함
  - 프로세서 내부에 있는 모든 회로가 다 동작할 수 있을만큼 Clock이 느려야만 정상적으로 동작함



중요한 Timing Parameter의 의미

- **Longest Propagation Delay**
  - Critical Path Delay라고 불리움
  - Critical Path: 프로세서 내 회로들 중 가장 오랜 시간이 걸리는 회로
  - Timing Analysis를 통해 확인할 수 있음

실행 시간

- 우리는 이제 CPU 실행 시간을, 어떤 프로그램이 수행되는 시간을 측정하기 위한 기준으로 활용할 것임
  - **Execution time = Clock cycles for program * Clock cycle time**
- Clock Cycle Time은 Clock Rate의 역수이므로
  - **Execution time = $\frac{\text{Clock cycles for program}}{\text{Clock rate}}$**

Clock Cycle의 측정

- **CPU Clock Cycles / Program (= 하나의 Program이 사용하는 CPU Clock Cycle의 수)는 직관적이지 않음**
  - 왜? 아키텍처/프로세서마다 Clock Cycle이 다를 수 있음 (e.g., single-cycle vs. multi-cycle)

- **CPI (Cycles Per Instruction)이 자주 사용됨**
  - 명령어마다 # of Cycle이 다를 수 있기 때문에 (e.g., single-cycle vs. multi-cycle), CPI는 평균 값을 계산함
  - CPI는 서로 다른 ISA를 비교하기 위해 사용할 수 있음
  - IPC (Instruction Per Cycle)는?

Using CPI

- 다음과 같이 다시 쓸 수 있음
  - Execution time = Instructions * CPI * Clock Cycle Time
    - **Execution time =** $\frac{\text{Clock cycles for program}}{\text{Clock rate}}$
- 성능 향상 (= 실행 시간 감소)는 Clock Rate의 증가 (= Clock Cycle Time의 감소), 낮은 CPI, 또는 명령어 수를 줄여서 달성할 수 있음
- 디자이너는 각 Cycle의 길이와 필요한 Cycle 수 사이의 균형을 잘 맞춰야 함

CPI의 예

- Machine A: 1ns Clock and CPI of 2.0
- Machine B: 2ns Clock and CPI of 1.2
- 어떤 Machine이 빠를까?

Example Solution

- 각 Machine의 CPU 시간 계산
  - **Execution time$_A$ = I * 2.0 * 1ns = 2.0 * I ns**
  - **Execution time$_B$ = I * 1.2 * 2ns = 2.4 * I ns**

- 성능 비교

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

- Machine A가 Machine B보다 1.2배 빠름
  - CPI가 언제나 정확한 성능 지표가 되지는 않음

CPI Variability

- 다른 종류의 명령어는 같은 프로세서에서 다른 Cycle 수를 가질 수 있음 (e.g., Multi-Cycle)
- CPI는 보통 명령어 종류에 따라 다르게 Report됨

  - **Clock cycles** = $\sum_{i=1}^{n} (CPI_i \times C_i)$

  - CPI_i : the CPI for the class of instructions
  - C_i: the count of that type of instructions

명령어 Mix에서의 CPI

- $CPI = \dfrac{\sum_{i=1}^{n} (CPI_i \times C_i)}{\text{Instruction Count}}$

- CPI Example

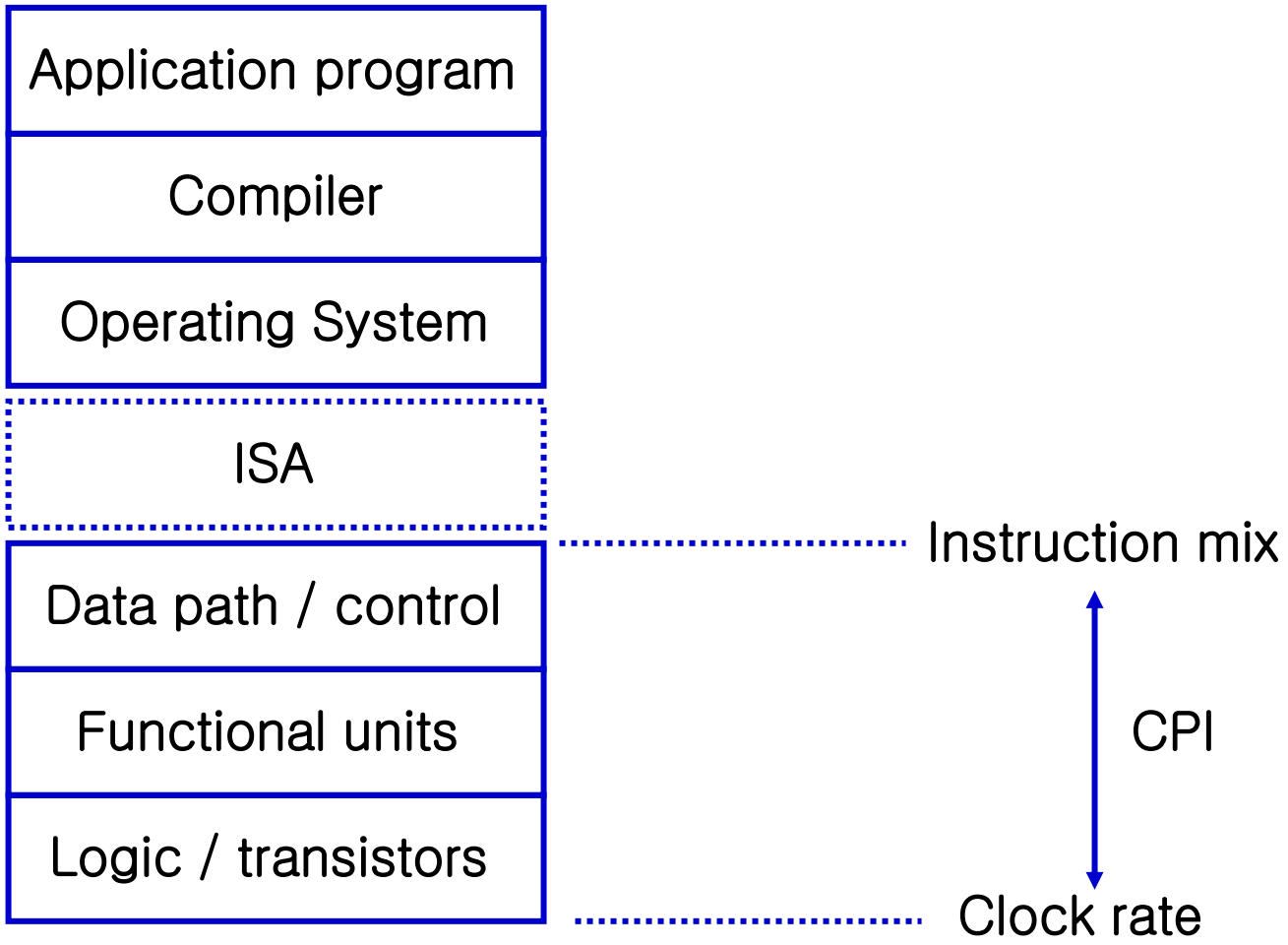
Instruction Class	Frequency	CPI_i
ALU operations	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

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

Tradeoffs

- 명령어 수, CPI, 그리고 Clock Rate 사이의 Tradeoff를 고려해야 함



CPU 성능의 측면

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

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

널리 사용되는 다른 성능 측정 기준

- MIPS (million instructions per second)
  - $\text{MIPS} = \dfrac{\text{Instruction count}}{\text{Execution time} \times 10^6}$
  - 문제
    - 명령어 Set에 대해 고려하지 않음
- MFLOPS (million floating-point operations per second)
  - 명령어 대신 Operation으로 성능을 측정함
  - E.g. 부동소수점 덧셈, 곱셈, 뺄셈, …

MIPS의 잘못된 사용

- Consider a 500MHz machine

Class	CPI
Class A (ALU)	1
Class B (Load/Store)	2
Class C (Branch)	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?

MIPS의 잘못된 사용 (I)

Clock Cycle의 계산

- Clock cycles = $\sum_{i=1}^{n} (CPI_i \times C_i)$

- Clock cycles$_{comp1}$ = (1*5M) + (2*1M) + (3*1M) = 10M
- Clock cycles$_{comp2}$ = (1*10M) + (2*1M) + (3*1M) = 15M

실행 시간의 계산

- Execution time = (Instruction count * CPI) / Clock rate
  = Clock cycles / Clock rate
- Execution time$_{comp1}$ = 10M/500M = 0.02sec
- Execution time$_{comp2}$ = 15M/500M = 0.03sec

Compiler1에서 생성한 코드가 1.5배 빠름

MIPS의 잘못된 사용 (II)

MIPS의 계산

- $\text{MIPS} = \dfrac{\text{Instruction count}}{\text{Execution time} \times 10^6}$

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

Compiler 2에서 생성한 코드가 빠름

- 잘못된 결론을 도출함

벤치마크

- **사용자는 종종 성능에 대한 지표를 원함 (e.g., 구매한 스마트폰/컴퓨터의 성능을 점수의 형태로 평가하고 싶어함)**
- **다양한 워크로드의 특성을 포함하는 벤치마크를 성능 평가를 위해서 사용함**
  - 실제 어플리케이션을 사용하면 좋지만, 굉장히 많은 종류의 어플리케이션이 존재하기 때문에, 이를 모두 사용해 평가하는 것은 불가능함
  - 대신, 실제 어플리케이션과 동일한 동작을 수행하는 명령어들 (또는 소스코드)로 구성된 벤치마크를 수행해, 점수 형식으로 성능을 평가함
- **다양한 워크로드의 특성을 포함하는 벤치마크를 성능 평가를 위해서 사용함**
  - Relevant: 타겟 시스템 (e.g., 모바일, 서버)에서 실제로 수행될법한 워크로드
  - Understandable: 결과를 쉽게 이해할 수 있어야 함
  - Good Metric(s): 사용자들은 보통 점수를 선호함
  - Scalable: 시스템 또는 어플리케이션이 달라져도 사용할 수 있어야 함
  - Coverage: 타겟 시스템에서 중요한 것을 Oversimplify하지 않아야 함
  - Acceptance: 판매자와 사용자 모두 '인정'해야함

Amdahl's Law

실행시간의 향상은 1) 얼마나 향상되었는지, 그리고 2) 향상된 부분이 전체에 얼마나 영향을 미치는지에 의해 결정됨

- **성능 향상 이후의 실행 시간의 계산**

$$= \left[\frac{\text{성능향상에 의해 영향을 받는 실행 시간}}{\text{성능 향상이 얼마나 되었는지}} + \text{성능 향상에 영향을 받지 않는 실행시간} \right]$$

- 또는

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

- 이 때, 하드웨어 및 실행 시간과 관련 없는 사항은 영향을 미치지 않음 (e.g., Code Size)

Amdahl's Law의 예시

- 부동소수점 명령어의 성능이 2배 향상되었음
- 그러나 부동소수점 명령어는 전체 명령어의 10%임

부동소수점 명령어들의 비율

부동소수점 명령어가 아닌 다른 명령어들의 비율

성능의 2배 향상 = 실행시간이 1/2로 줄어듬

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

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

성능은 실행 시간의 역수
(5 ~ 6 Page 참고)

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