

How does your computer work?

Algorithm ↗ 작동이 일어나게 하는 단계적 집합
↓ 연산, 데이터 전행, 자동화된 추론 등을 수행

High Level Language : 고급언어 / 사람 중심 언어
↓ Compiler Machine Independent

Assembly Language : 저급언어
↓ Assembler 기계어와 1:1 대응되는 기호로 이루어진 언어

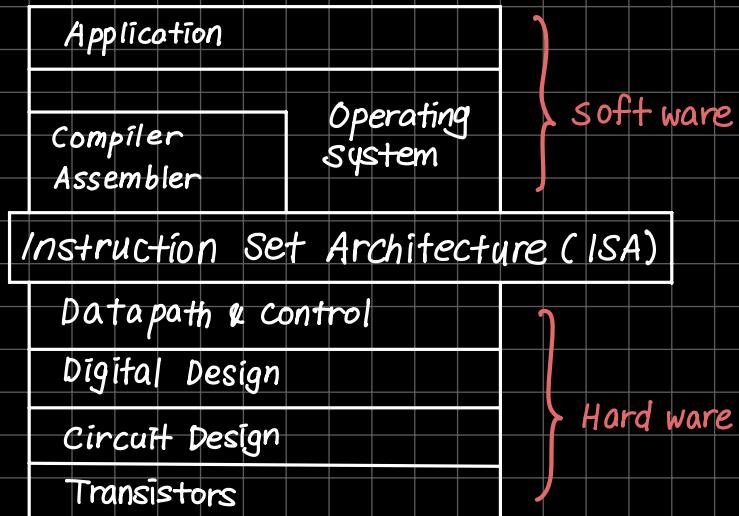
Machine Language : 저급언어 (기계어) Machine Dependent
↓ 컴퓨터가 직접 이해할 수 있는 언어 / 0과 1의 이진수 형태
수행시간 빠름

Hard Disk : 보조기억장치 / 비휘발성 메모리
↓ 사용자의 데이터와 프로그램 저장

Main Memory : 주기억장치 (=RAM) / 휘발성 메모리

Processor (CPU/GPU)

Computer System Organization



Computers Are Pervasive

- Computers in Automobiles
- Cellphones
- Human Genome Project
- World Wide Web
- Search Engine

컴퓨터 응용 분야의 종류와 특성 Classes of Computers

- 개인용 컴퓨터 (PC) : 낮은 가격, 좋은 성능 / 소프트웨어 실행
 - General Purpose, variety of software
 - Subject to cost/performance tradeoff
- 서버 : 여러 사용자, 대형 프로그램
 - High Capacity, performance, reliability
 - Range from small servers to building sized
- 슈퍼 컴퓨터 (서버의 한 종류) : 과학·공학 계산 / 컴퓨터 시장의 작은 부분만을 차지
 - High end scientific and engineering calculations
 - Highest Capability / represent only a small fraction of the overall computer market
- 임베디드 컴퓨터 : 장치에 포함된 숨겨진 컴퓨터 / 엄격하게 제한된 가격·성능·전력소모
 - Hidden as components of system
 - Stringent power/ performance / cost constraints → 가장 중요한 것은 배터리!

All the Common Size Terms

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%
ronnabyte	RB	10^{27}	robbibyte	RiB	2^{90}	24%
queccabyte	QB	10^{30}	queebibyte	QiB	2^{100}	27%

from 2022 (사용예정)

digits 커질수록 Decimal과 Binary 차이 커진다.

Post PC Era

- Cellphone (excluding smartphones) ↓
- Smartphone ↑
- Tablet ↑
- PC (excluding tablets) -

What you will learn?

- 고급언어로 작성된 프로그램 → 기계어 번역 과정
하드웨어는 프로그램을 어떻게 실행하는가?
- SW와 HW 사이의 인터페이스
- 프로그램 성능을 결정하는 요소들 / 프로그래머는 성능을 어떻게 개선할 수 있는가?
- 성능 개선을 위해 하드웨어 디자이너들은 어떤 기술을 사용할 수 있는가?
- 병렬처리 (Parallel Processing) 란 무엇인가?

프로그램 성능 (Performance)의 이해

- Algorithm : determines number of operation executed
소스 프로그램 문장 수와 입출력 작업 수 결정
- Programming Language, Compiler, Architecture
 - : determine number of machine instructions executed per operation
각 소스 프로그램 문장에 해당하는 기계어 명령 수 결정
- Processor and Memory System
 - : determine how fast instructions are executed
명령어의 실행 속도 결정
- I/O System : determines how fast I/O operations are executed
(including OS) 입출력 작업의 실행속도 결정

Seven? Eight? Great Ideas

① 설계를 단순화하는 추상화 Use Abstraction to Simplify Design

하위수준의 상세한 기술을 보이지 않게 → 상위수준의 모델을 단순화

② 자주 생기는 일을 빠르게 Make Common Case Fast

③ 병렬성을 통한 성능개선 Performance via Parallelism

④ 파이프라인을 통한 성능개선 Performance via Pipelining

병렬성의 특별한 형태 ... 돌림노래 구조와 비슷

⑤ 예측을 통한 성능개선 Performance via Prediction

⑥ 메모리 계층구조 Hierarchy of Memories

최상위 계층 → 비트당 가격이 비싸고 작고 빠른 메모리
최하위 계층 → 느리지만 크고 비트당 가격싼 메모리) cache 와 관련...

⑦ 여유분을 이용한 신용도 개선 Dependability via Redundancy

모든 물리소자는 장애가 발생할 수 있으므로 여유분이 있으면 신용도↑

⑧ Design for Moore's Law

18 ~ 24개월마다 반도체에 접착되는 소자의 수가 2배씩 증가
매년 향상되는 컴퓨팅의 성능을 고려하여 컴퓨터를 설계해야 한다
⇒ 현재는 유효하지 않은 법칙 . 트랜지스터의 증가 속도가 감소하고 있다

Below Your Program

내려갈수록 컴퓨터의 안쪽

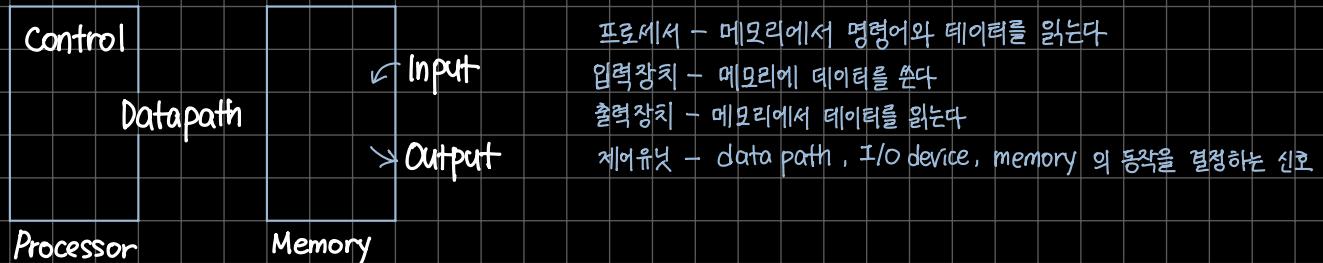
- Application Software : written in High Level Language (HLL)
- System Software
 - Compiler : translate HLL Code → Machine code
 - Operation System (OS) : service code
 - Handling input/output 기본적 입출력 작업의 처리
 - Managing memory and storage 보조기기장치 및 메모리 할당
 - Scheduling Tasks & sharing resources
- Hardware : processor, I/O controllers, memory

Levels of Program Code

- High-Level Language (HLL)
 - level of abstraction closer to problem domain
 - provides for productivity + portability
- Assembly Language
 - Textual representation of instructions
- Hardware representation
 - Binary digits (=bits)
 - Encoded instructions and data

Components of a Computer

- 컴퓨터의 5대 구성요소 : 입력 출력 메모리 Datapath 제어유닛
 - Same components for all kinds of computer - desktop, server, embeded ...



- Input / Output includes
 - User interface devices : Display, Keyboard, Mouse
 - Storage devices : Hard disk, CD/DVD, Flash
 - Network adapters : For communicating with other computers

Display : LCD Screens

- picture elements (=pixels)
- mirrors content of frame buffer memory

그래픽 하드웨어는 스크린에 표시될 화상을 frame buffer에 저장하였다가

기억된 화소의 비트패턴을 재생속도에 맞추어 그래픽 디스플레이로 보낸다

frame buffer의 각 좌표값이 raster scan CRT 디스플레이에서 해당좌표의 밝기를 결정한다

Touch Screen

- Post PC device
- Supersedes keyboard & mouse
- Resistive and Capacitive types
 - Most tablets, smart phones → capacitive
 - Capacitive : allows multiple touches simultaneously

Opening the Box

- 집적회로칩 (integrated circuit, IC) : 수천만 개의 트랜지스터가 결합된 부품
- CPU (= 프로세서) : data path + control unit
 - 연산 수행, 입출력 장치에 신호를 보내 작동을 지시
 - ↳ data path : 산술 연산 수행
 - ↳ control unit : 작동지시
- 메모리 : 실행 중인 프로그램과 프로그램 실행에 필요한 데이터의 저장소
 - ↳ DRAM (Dynamic Random Access Memory)

Inside the Processor (CPU)

- Data path : performs operations for the data
- Control : sequences data path, memory, and more
- Cache Memory : Small fast SRAM memory for immediate access to data

Abstractions 추상화

- Abstractions helps us deal with Complexity - Hide lower level detail
- * · Instruction Set Architecture (ISA) - Hardware / Software interface
 - 명령어, 레지스터, 메모리 접근, 입출력 등을 포함하여 기계어 프로그램을 작성하기 위해서 알아야하는 모든 정보
- ABI (Application Binary Interface) - ISA + OS interface
 - 응용 프로그램과 운영체제, 응용 프로그램과 라이브러리 사이에 필요한 저수준 인터페이스
- * · Implementation 구현 - the details underlying and interface
 - 구조 추상화를 준수하는 하드웨어
 - 규격서, 알고리즘을 프로그래밍 OR 소프트웨어 배치를 통해 구현하는 것

A Safe Place for Data

- 휘발성 메모리 volatile main memory - loses instructions and data when power off
- 비휘발성 메모리 non-volatile secondary memory - magnetic disk, flash memory, optical disk (CD ROM/DVD)

Networks

- Communication, resource sharing, non-local access
- 근거리 네트워크 Local Area Network (LAN) - ex) Ethernet
- 원거리 네트워크 Wide Area Network (WAN) - ex) the Internet
- Wireless Network - Wifi, Bluetooth

Technology Trends

- Electronics technology continues to evolve
capacity, performance↑ / cost↓

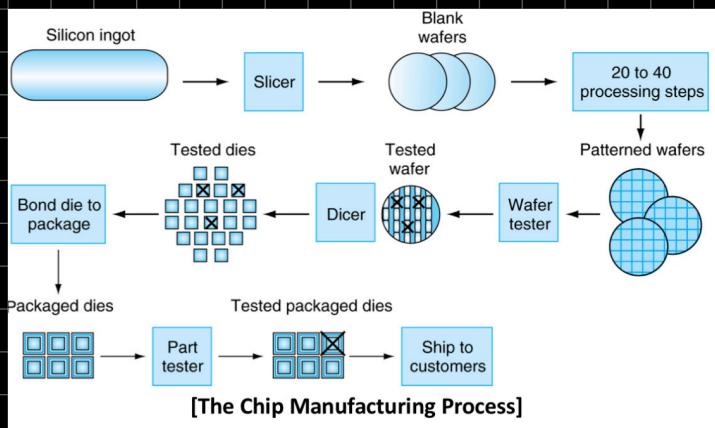
→ 반도체

Semiconductor Technology & Manufacturing ICs

- Silicon → 반도체 천연원소
- 불순물을 증가하여 실리콘의 작은부분을 바꿀 수 있다
 - 전기의 양도체 conductors
 - 전기의 절연체 insulators
 - 조건에 따라 도체/ 절연체 switch

- Yield : proportion of working dies per wafer
수율 전체 다이 중 정상다이 비율

The Chip Manufacturing Process



* die : 집적회로
(칩) 웨이퍼에서 잘라낸 개개의 사각형
여러조각으로 나누면 결함 발생 시
웨이퍼 전체를 버리는 게 아니라
해당 칩만 버리면 된다

Integrated Circuit Cost

$$\cdot \text{Cost per die} = \frac{\text{웨이퍼당 가격}}{\text{다이 원가}} = \frac{\text{Cost per Wafer}}{\text{Dies per Wafer} \times \text{Yield}}$$

↗ 즉 웨이퍼당 가격 / 정상 다이 수
웨이퍼 당 데이위드 수 수율

$$\cdot \text{Dies per Wafer} \approx \frac{\text{Wafer Area}}{\text{Die Area}}$$

↗ 웨이퍼는 둥글고 다이는 사각형이라서
근사식이다

$$\cdot \text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area})^N)}$$

면적당 결함의 수

- 즉, 원가는 다이·웨이퍼 면적이나 결함비율과 non-linear relation
 - Wafer cost, area → fixed
 - defect rate → determined by manufacturing process
 - Die area → determined by Architecture and Circuit Design

성능의 정의

→ 관점에 따라 여러가지로 정의할 수 있다

- Response Time 응답시간 = execution time 실행시간
How long it takes to do a task 작업 개시에서 종료까지의 시간
↳ 개선) replacing the processor with a faster version

- Throughput 처리량 = 대역폭 (band width)
Total task done per unit time
↳ 개선) Adding more processors

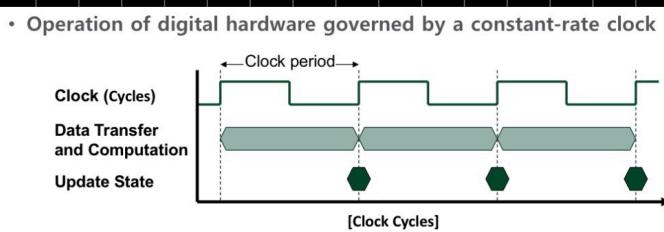
Relative Performance

- 성능 = $1/\text{실행시간}$
- "X is faster than Y" → $\frac{\text{성능 } X}{\text{성능 } Y} = \frac{\text{실행시간 } Y}{\text{실행시간 } X} = n$
 $n > 1 \rightarrow X \text{ 가 } Y \text{ 보다 } n \text{ 배 빠르다}$

Measuring Execution Time

- Elapsed Time 경과시간 응답시간 벽시계시간 (wall clock time)
 - : total response time, including all aspects
한 작업을 끝내는데 필요한 전체시간 - 디스크 접근, 메모리 접근, 입출력 작업, OS 오버헤드, idle time ...
- Determines System performance
- CPU Execution Time (CPU Time)
 - : time spent processing a given job
↳ I/O time, 다른 프로그램 실행시간 제외
 - ↙ user CPU time 사용자 CPU 시간 - 사용자 프로그램 실행
 - System CPU time 시스템 CPU 시간 - OS의 작업 수행
 - Different programs are affected differently by CPU and system performance
 - 사용자가 느끼는 응답시간은 CPU time 아니라 execution time

CPU Clocking



- Clock period : duration of clock time
클럭 주기
rising edge ~ rising edge
ex) $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- clock frequency (rate) : cycles per sec
클럭 속도
ex) $40\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$
- $\text{clock rate} = 1/\text{clock period}$
↳ 1초당 cycle ↑ ↳ 1 cycle 당 소요시간
 $1\text{GHz CR} = 1/\text{1ns CC (clock cycle period)} \quad n=10^{-9}, f=10^9$

CPU Time

$$\begin{aligned} &= \text{CPU Clock Cycles} \times \text{CPU Clock Time} \\ &\quad \text{클럭 사이클 수} \quad \text{1 클럭 사이클 시간} = \text{클럭 주기 clock period} \\ &= \text{CPU Clock Cycles} / \text{Clock Rate} \end{aligned}$$

Performance improved by ...

- number of clock cycles ↓
- clock rate ↑
- Hardware designer must often trade off clock rate against clock count
 $\text{clock time} = \text{clock count} = \text{clock period}$

CPU Time 예제

- Computer A : 2GHz clock . 10s CPU Time
- Computer B : Target - 6s CPU Time
clock rate can be improved
but requires 1.2 times more clock cycles than A

$$\begin{aligned} \text{sol) CPU Time A} &= \text{CPU Clock Cycles A} / \text{CPU Clock Rate A} \\ 10\text{s} &= \text{CPU Clock Cycles A} / 2 \times 10^9 \text{Hz} \\ \text{CPU Clock Cycles A} &= 2 \times 10^{10} \end{aligned}$$

$$\begin{aligned} \text{CPU Time B} &= \text{CPU Clock Cycles B} / \text{CPU Clock Rate B} \\ 6\text{s} &= 1.2 \times (2 \times 10^{10}) / \text{CPU Clock Rate B} \\ \text{CPU Clock Rate B} &= 4 \times 10^9 \text{Hz} = 4 \text{GHz} \end{aligned}$$

결론: B의 CPU 속도는 A보다 2배 빨라야 한다

Instruction Count and CPI

- CPU clock cycles = Instruction Count × Cycles per Instruction (CPI)
 $\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$
cf) $\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock cycle Time}$
 $= \text{Instruction Count} \times \text{CPI} / \text{Clock Rate}$
- 명령어마다 실행시간 다르므로 CPI는 프로그램이 실행한 모든 명령어에 대한 평균 cycle 수를 사용
→ CPU Hardware에 의해 결정된다
- 명령어 집합구조 같으면 프로그램에 필요한 명령어 수 같다
 $\text{Instruction Count for a program} \rightarrow \text{ISA, program, compiler에 의해 결정}$

CPI 예제

- Computer A : Cycle Time 250ps, CPI = 2.0 for program X
 - Computer B : Cycle Time 500ps, CPI = 1.2 for the same program) same ISA
- Q. Which computer is faster for a program X and by how much?

Sol) A와 B의 CPU Time을 비교해야 한다.

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

같은 프로그램, ISA \rightarrow Instruction Count 같다

$$\text{CPU Time A} = I \times 2.0 \times 250\text{ps}$$

$$\text{CPU Time B} = I \times 1.2 \times 500\text{ps}$$

$$\frac{\text{CPU Performance A}}{\text{CPU Performance B}} = \frac{\text{CPU Time B}}{\text{CPU Time A}} = \frac{I \times 1.2 \times 500\text{ps}}{I \times 2.0 \times 250\text{ps}} = 1.2$$

결론: A가 B보다 1.2배 더 빠르다

CPU Time과 성능 \rightarrow 역수 관계 주의!!

CPI in more details

- diff. instruction classes \rightarrow diff. numbers of cycle

$$\text{Clock Cycles} = \sum (\text{CPI} \times \text{Instruction Count})$$

- 가중평균 CPI (Weighted average CPI)

$$\text{CPI} = \text{Clock Cycles} / \text{Instruction Count}$$

$$= \sum (\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count total}})$$

↳ relative frequency (상대빈도)

코드의 비교 예제

- 컴파일러 설계자가 두 코드 중 하나를 선택하려 한다.

	A	B	C
CPI	1	2	3
Instruction Counts	code 1	2	1
code 2	4	1	1

* A, B, C는 명령어 유형

- Q. Which code sequence executes the most instructions?

Which will be faster?

What is CPI for each sequence?

Sol) (a) code 1 : $2 + 1 + 2 = 5$ 개 명령어

code 2 : $4 + 1 + 1 = 6$ 개 명령어

(b) Clock cycle 1 : $1 \times 2 + 2 \times 1 + 3 \times 2 = 10$ cycles

Clock Cycle 2 : $1 \times 4 + 2 \times 1 + 3 \times 1 = 9$ cycles

코드 2의 실행 속도가 더 빠르다

(c) 가중평균 CPI 구하기

$$\text{Weighted average CPI (1)} = 10 / 5 = 2$$

$$\text{Weighted average CPI (2)} = 9 / 6 = 1.5$$

$\Rightarrow \text{CPI} = \text{Instruction당 cycle}$

즉, CPI 작아야 더 많이 실행할 수 있다

Performance Summary

$$\begin{aligned} \text{Time} &= \text{seconds / program} \\ &= \frac{\# \text{ of instructions}}{\text{program}} \times \frac{\text{Clock cycles}}{\# \text{ of instructions}} \times \frac{\text{seconds}}{\text{clock cycles}} \end{aligned}$$

☆☆

- performance depends on :
 - ① Algorithm → Instruction Counts, CPI
 - ② Programming Language → Instruction Counts, CPI
 - ③ Compiler → Instruction Counts, CPI
 - ④ ISA → Clock Rate, Instruction Counts, CPI

Units of measure

- CPU Execution Time for a program : sec. for the program
- Instruction Count : Instruction executed for the program
- Clock Cycles per Instruction (CPI) : Average number of clock cycles per instruction
- Clock Cycle Time : sec. per clock cycle (clock period)

Always bear in mind that the only complete and reliable measure of computer performance is **time**

Power Trends

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

~ 공식 외울 필요X
30배↑ 5V → 1V 1000 배↑
전압 낮아져서 주기 1000배 빨라져도 Power 30배만 증가
유니코어 시스템의 한계 → power (전력소모, 발열) 때문에 frequency 성능 발전 한계
Constrained by **power**, Instruction level parallelism, memory latency

Multi Processors

- requires explicitly parallel programming
기존의 sequential programming 으로는 multicore 환경에서의 이점X
- Hard to do ① programming for performance
 - ② Load Balancing 프로세서 일 배분
 - ③ Optimizing communication & Synchronization
- Compared to instruction level parallelism
 - ① hidden from the programmer - 추상화, 신경 쓸 필요X
 - ② hardware executes multiple instructions at once

Pitfall : Amdahl's Law

함정 : 컴퓨터의 한 부분만 개선하고 그 개선됨 양에 비례해서 전체 성능이 좋아지리라고 기대하는 것

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{Improvement Factor}} + T_{\text{unaffected}}$$

ex) 전체 100s 中 러닝타임 80s 소요될 때
러닝타임 성능을 얼마나 향상시켜야 전체 성능이 5배 증가하는가?

$$20 = \frac{80}{n} + 20$$

$\hookrightarrow n$ 이 4가 될 수 없으므로 불가능

Corollary : make the common case fast

Fallacy : Low Power at Idle

오류 : 이용률 낮은 컴퓨터는 전력소모가 적다

- i7 power benchmark
 - at 100% load : 258W
 - at 50% load : 176W (66%)
 - at 10% load : 121W (47%)
- Google Data Center
 - mostly operates 10 ~ 50% load

프로세서 설계 시 load \propto power 이뤄질 수 있도록 하는 것이 이상적이다

Pitfall : MIPS as a Performance Metric

오류 : 성능식의 일부분을 성능의 척도로 사용하는 것

- MIPS : Millions of Instructions per second 명령어 개수 / (실행시간 $\times 10^6$)
- MIPS는 단순히 명령어 실행 속도를 나타내며 명령어 하나가 수행할 수 있는 일의 양은 반영 X
- 같은 CPU에서도 실행 프로그램에 따라 달라진다
 - = CPI는 CPU에 따라서도 달라지지만 프로그램에 따라서도 달라진다
- MIPS는 ① ISA의 차이 ② 명령어의 차이 ③ 프로그램의 차이를 반영하지 못한다

$$\text{MIPS} = IC / (\text{Execution Time} \times 10^6)$$

$$\begin{aligned} &= \frac{IC}{(IC \times CPI) / \text{Clock Rate} \times 10^6} \\ &= \text{Clock Rate} / (CPI \times 10^6) \end{aligned}$$

\hookrightarrow CPI의 영향 받는다

Conclusion

- ① 기술의 발전 → Performance / Cost 증가
- ② Hierarchical layers of Abstraction in both SW & HW
- ③ ISA - HW & SW Interface
- ④ Execution Time – the best performance measure
- ⑤ Power = limiting Factor
 - use parallelism to improve performance