Chapter1. Computer Abstractions and Technology

wnwlcks123 · 2022년 8월 2일

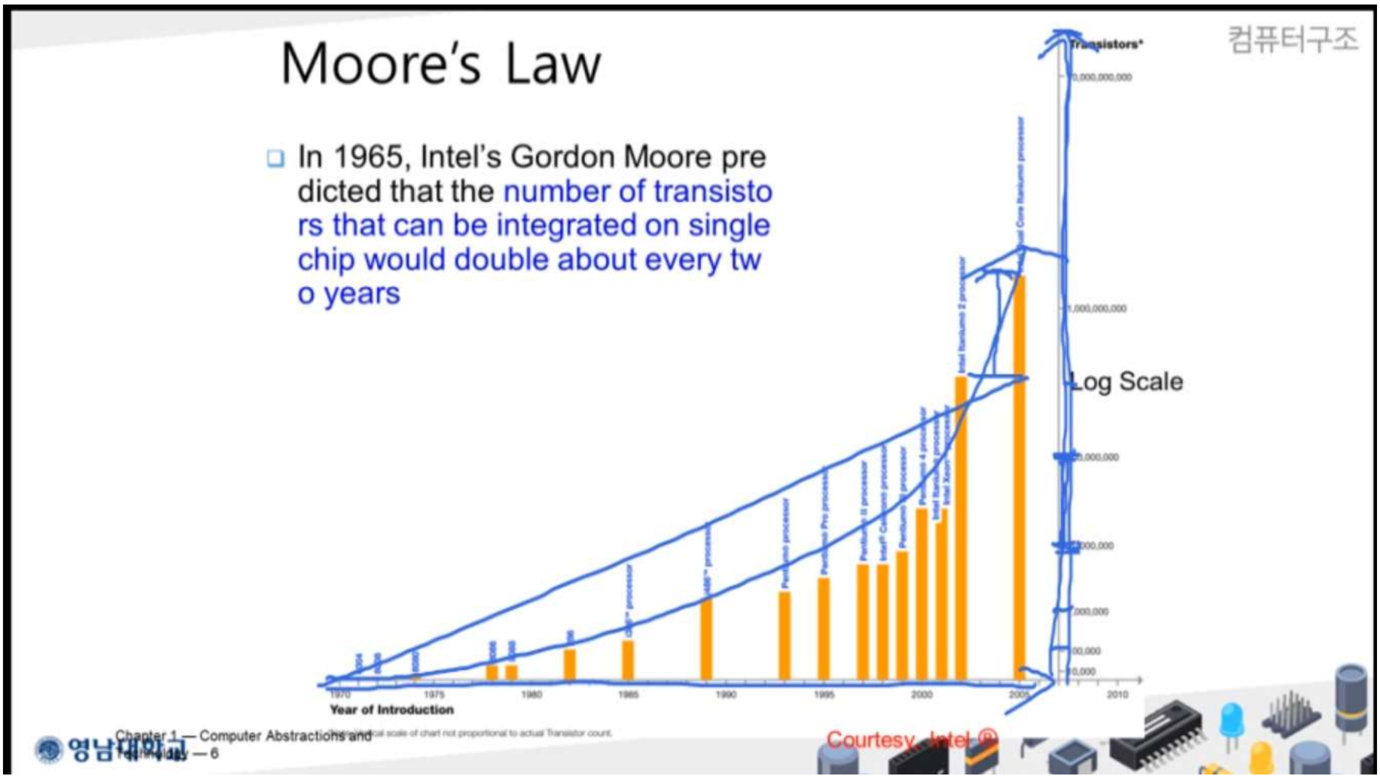
http://www.kocw.net/home/m/cview.do?cid=16bd07027739ad22

무어의 법칙 - 매 2년마다 트랜지스터의 수가 두배씩 증가한다고 예측한다

하지만 실제 트랜지스터는 고든무어가 예측한것보다 더 증가한다.

로그스케일 : 10 100 1000 10000단위로 증가하는것.

실제로는 로그스케일처럼 증가했다.



현재는 무어의 법칙은 폐기 하나의 칩에 들어가는 트랜지스터의 수를 제한함.

2000년대 초기까지는 맞는 법칙이었음.

현재는 많은 반도체 회사들이 무어의 법칙은 유효하지 않다고 선언하고있음.

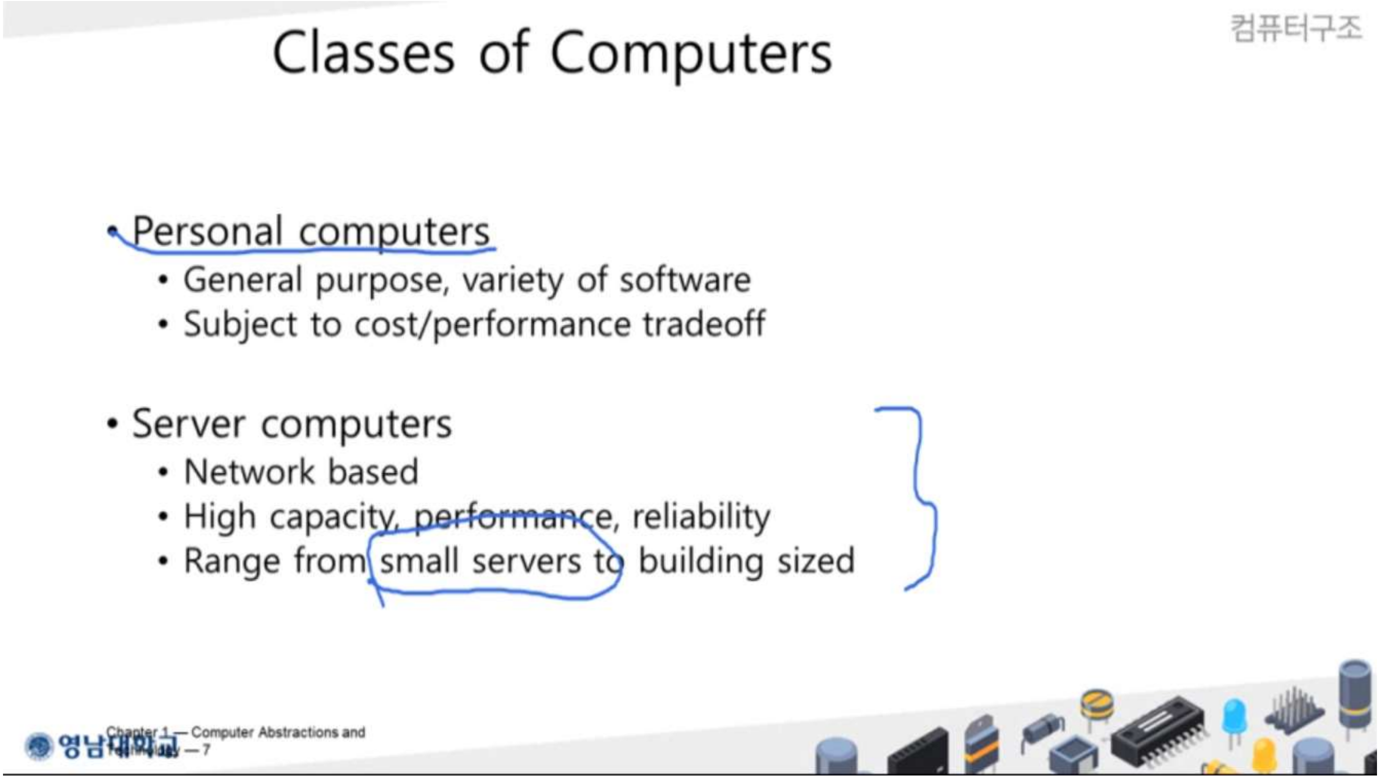
컴퓨터의 종류

Personal computers

- 랩탑, 데스크탑따위

Server computers

- 고사양 고성능 컴퓨터

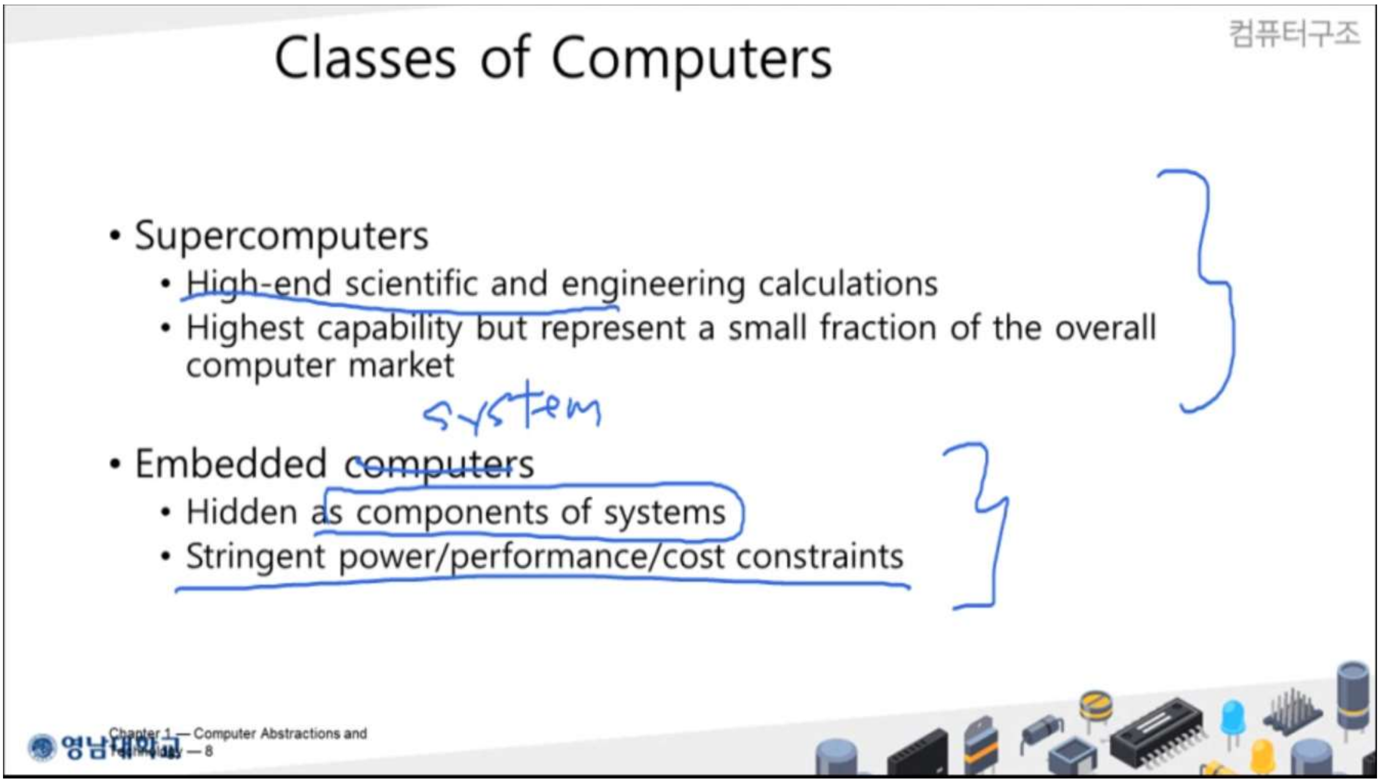


Supercomputers

- 아주 고가이고 특정한 목적에의한 컴퓨터

Embedded computers 혹은 Embedded computer

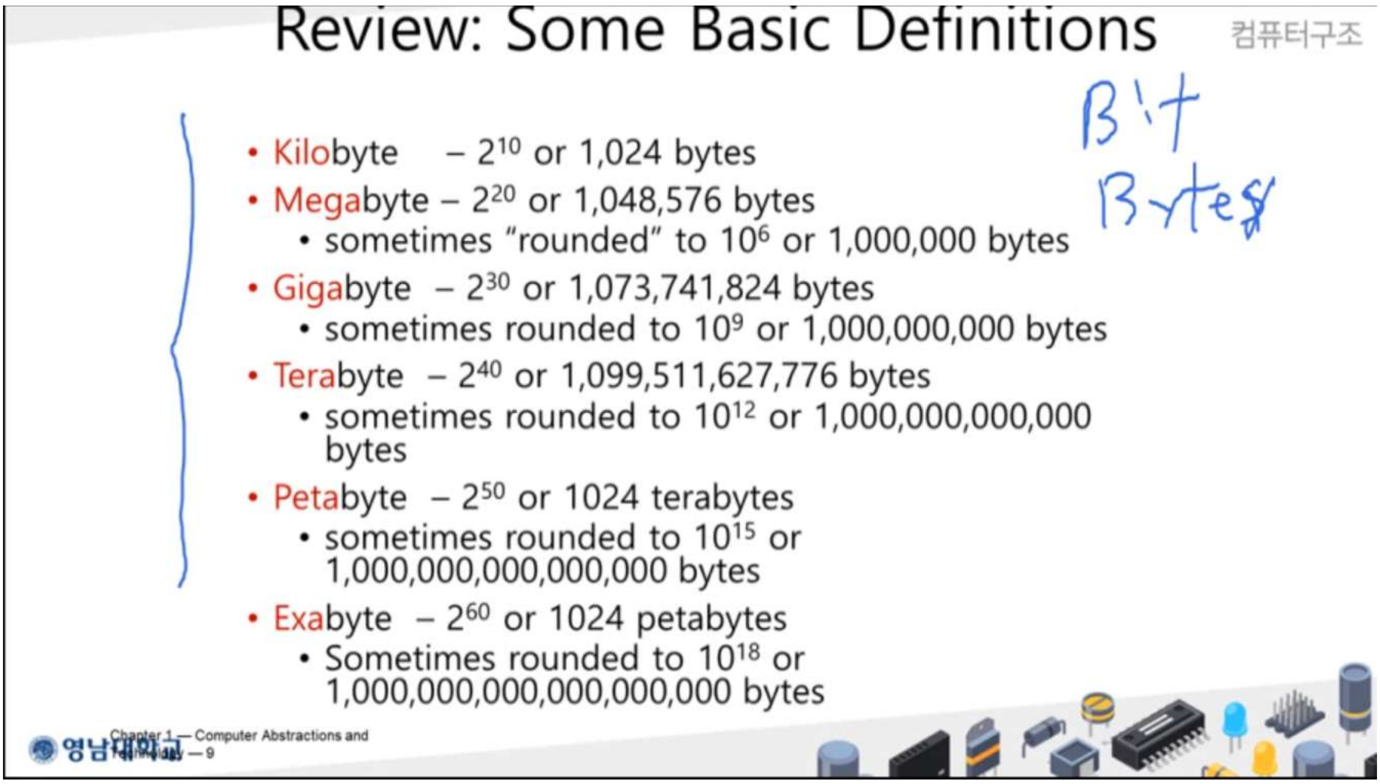
- 적은파워,적은성능,저비용
- 자동차에 들어가는 컴퓨터 따위



컴퓨터의 단위

Bit - 0또는 1

Byte - 8개의 Bit



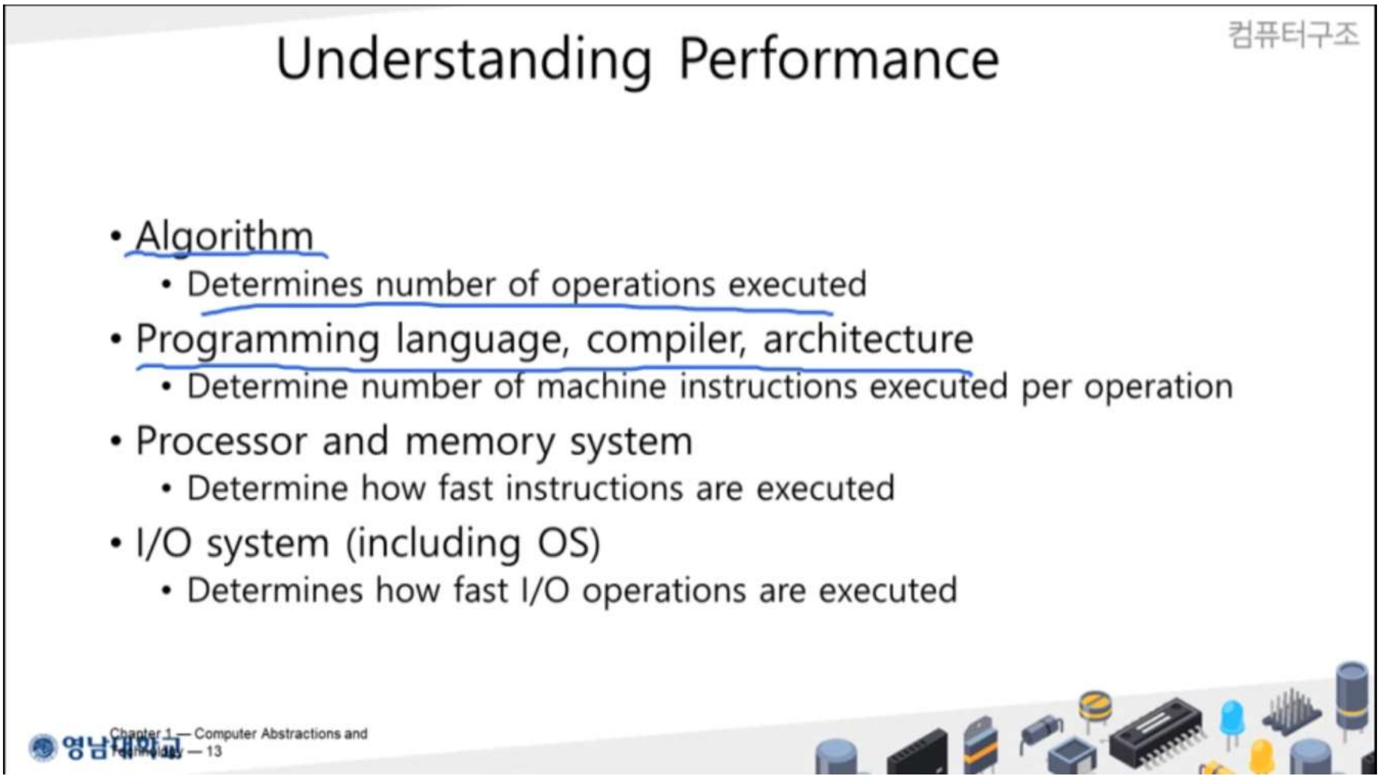
때때로 10의 n승으로 계산이 될 때도 있다.(분야마다 다르다 ex:통신)

Personal Mobile Device (PMD)

- 스마트폰이나 태블릿

Cloud computing

- 아마존이나 구글등이 제공하는 서비스



퍼포먼스

Algorithm

- 오퍼레이션의 수를 줄이는 것.

Programming language, compiler, architecture

- 실행하는 명령의 수를 결정

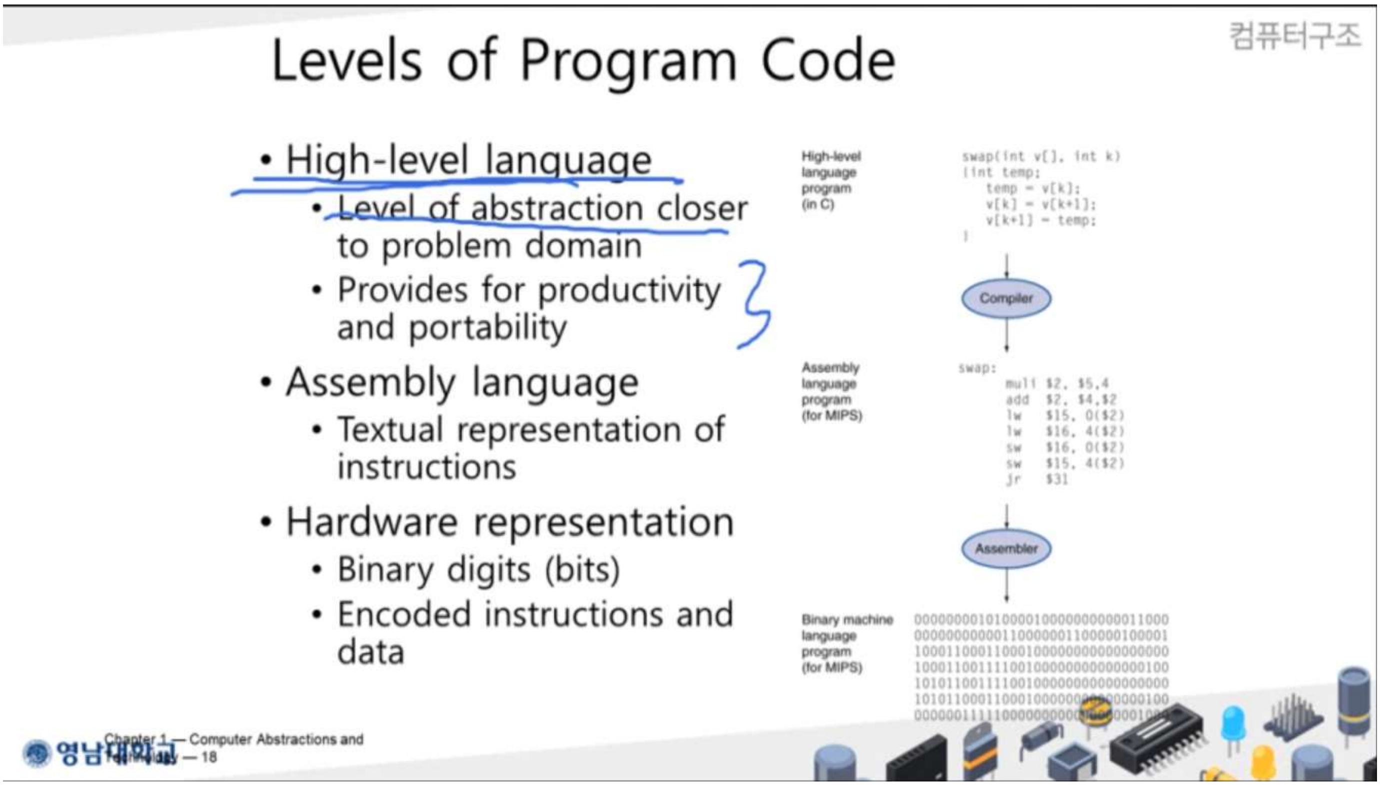
Processor and memory system

- 이 명령어들을 얼마나 빨리 실행 시키는가

I/O system (including OS)

- I/O오퍼레이션을 얼마나 빨리 처리하는가

컴퓨터 아키텍쳐를 발전시킨 중요한 8가지들



사람이 이해하기 쉬울수록 High-level language

Assembly language - 프로그래밍하기 아주 어려움

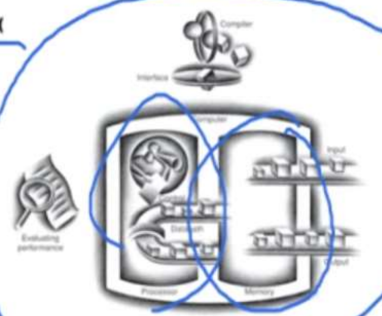
Hardwawre representation - 사람이 이해하기도 어려움

컴퓨터구조

Components of a Computer

The BIG Picture

- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers



컴퓨터구조

Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...

4장

- Cache memory
  - Small fast SRAM memory for immediate access to data

5장

Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

A Safe Place for Data

- Volatile main memory (DRAM)
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flas'
  - Opt

Volatile : 휘발성

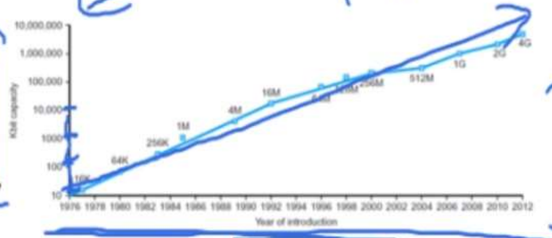
실제 칩을 만들때 어떤 소재를 사용하는가

Technology Trends

log-scale

Moore's Law

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

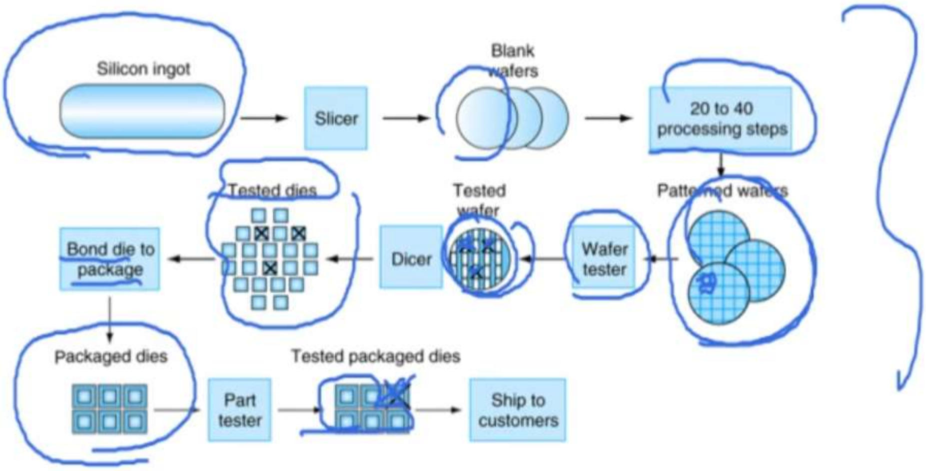
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

실제 칩을 만들때 어떤 소재를 사용하는가

Manufacturing ICs



공정

- Yield: proportion of working dies per wafer 수율

퍼포먼스

Response Time and Throughput

- Response time (latency)
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

컴퓨터 구조에서 퍼포먼스는

Response time과 Throughpu 을 대표적으로 말한다.

Response Time and Throughput

- Response time (latency)
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

response time ↓
throughput ↑

Throughput ↑

앞으로는 어떻게 하면 response time 을 줄일것인가가 퍼포먼스의 정의다.

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

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

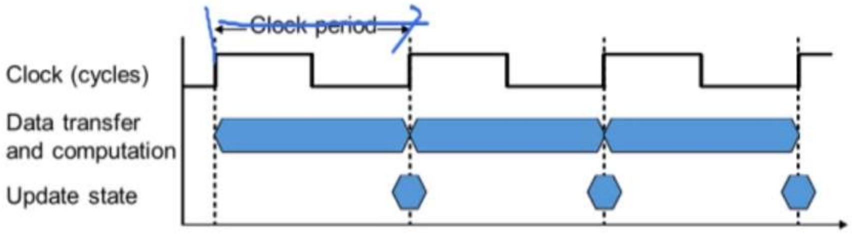
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

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

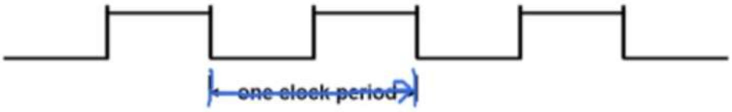


- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

Review: Machine Clock Rate

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$



10 nsec clock cycle => 100 MHz clock rate

5 nsec clock cycle => 200 MHz clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec (10^{-9}) clock cycle => 1 GHz (10^{9}) clock rate

500 psec clock cycle => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

퀴즈

아래의 중에서 성능이 향상되지 않는 경우는? (복수 선택 가능)

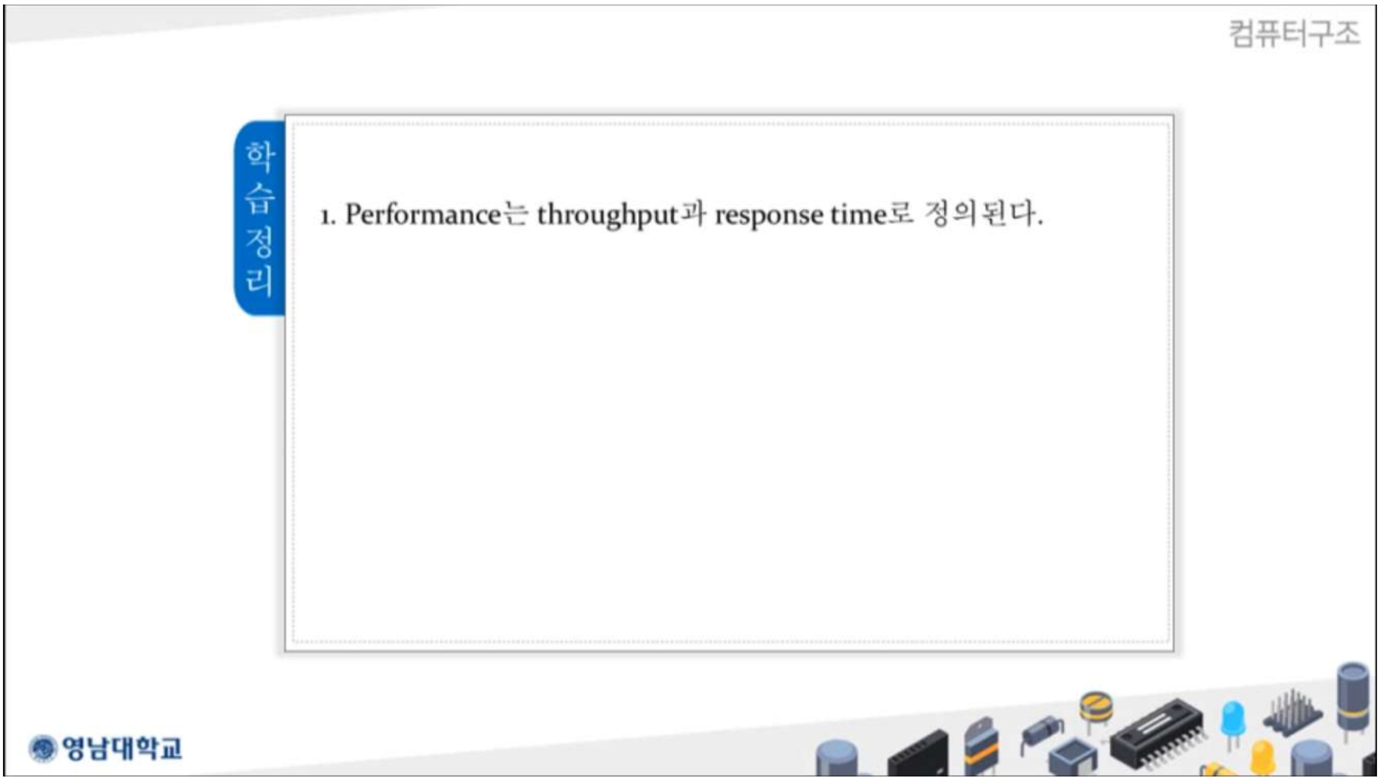
① Throughput이 증가하는 경우

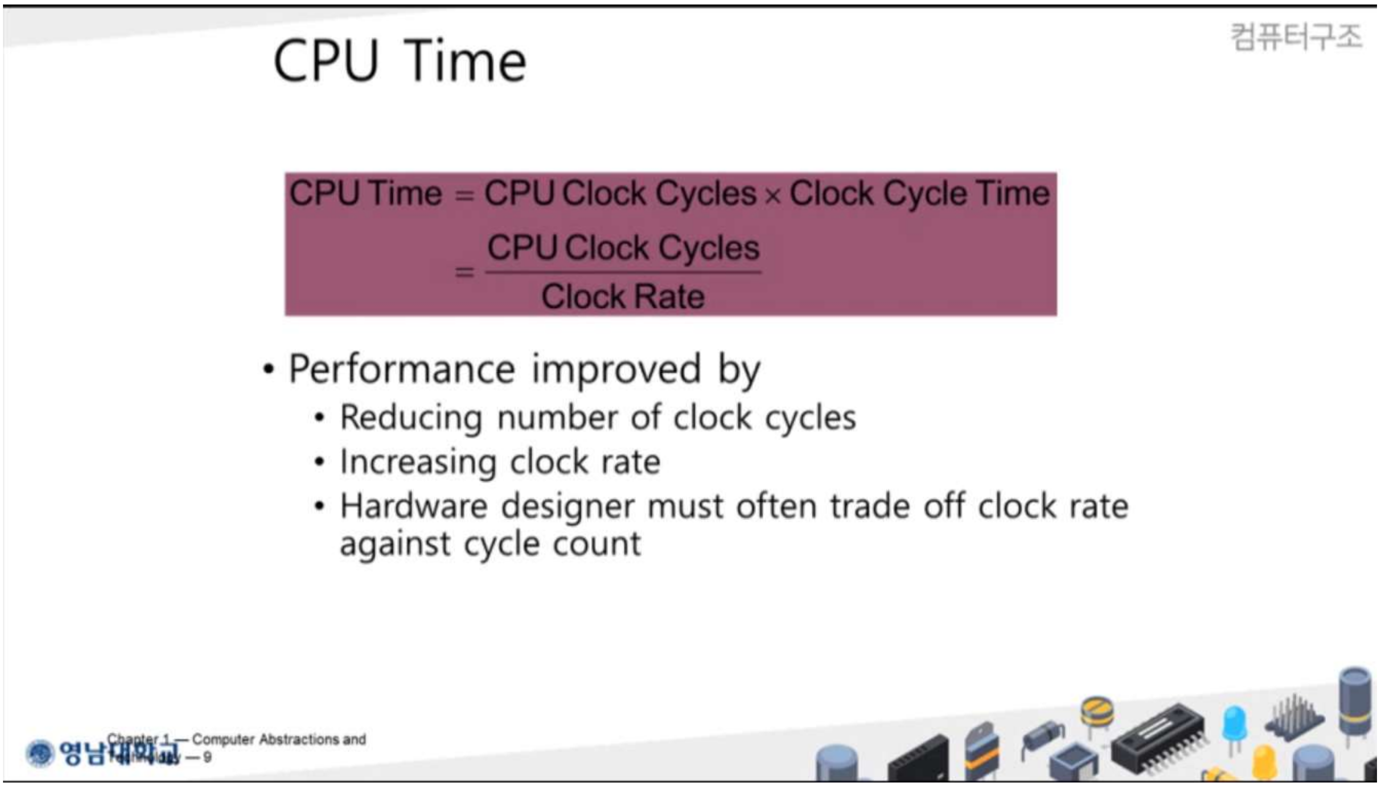
② Response time이 증가하는 경우

③ Throughput이 감소하는 경우

④ Response time이 감소하는 경우

정답 : 2,3





CPU 타임 = CPU 클락 사이클 x 클락 사이클 타임

= CPU 클락 사이클 / 클락 레이트

CPU 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?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10s \times 2GHz = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

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

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Cycles per Instruction = 명령어를 실행시키는데 시간이 얼마나 걸리는가

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?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps} \quad \leftarrow \text{A is faster…} \\ \text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2 \quad \leftarrow \text{…by this much}$$

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

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

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

A Simple Example

0.5

Op	Freq	CPI$_i$	Freq x CPI$_i$		
ALU	50%	.5 1	.5	.5	.25
Load	20%	1.0 5	.4	1.0	1.0
Store	10%	.3 3	.3	.3	.3
Branch	20%	.4 2	.4	.2	.4
		2.2	Σ = 1.6	2.0	1.95

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
  CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster
- How does this compare with using branch prediction to shave a cycle off the branch time?
  CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster
- What if two ALU instructions could be executed at once?
  CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

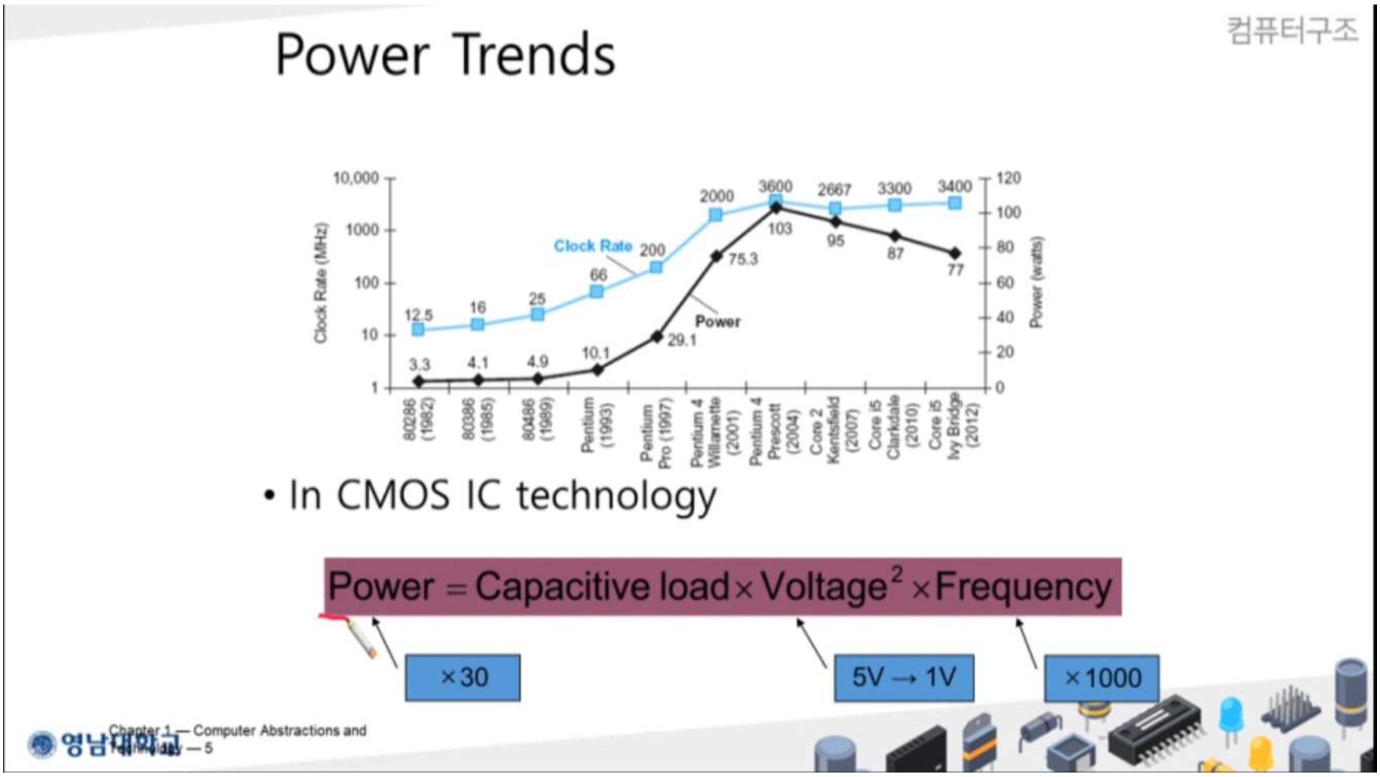
퀴즈

Clock period의 정의는 ?

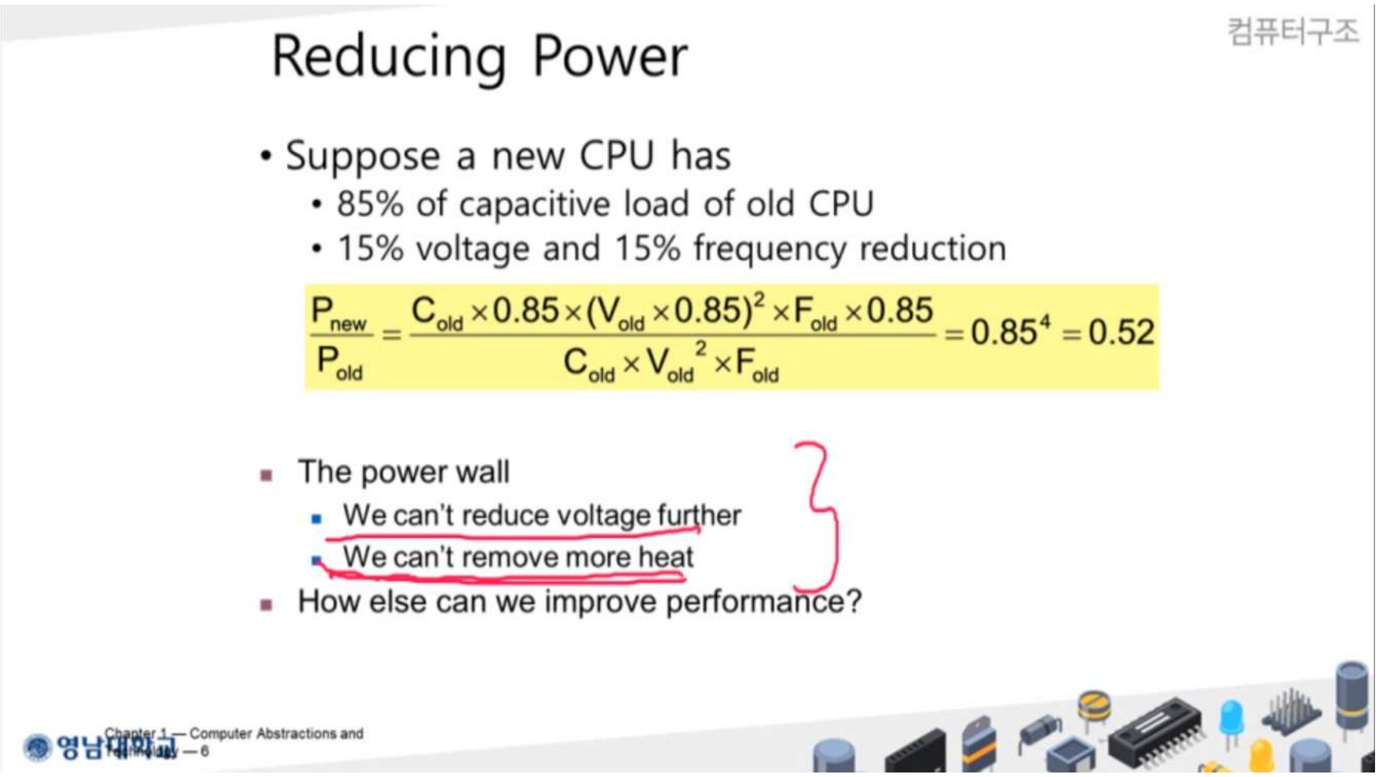
1/clock rate

2주 2차시로 넘어갑니다.

Power Wall

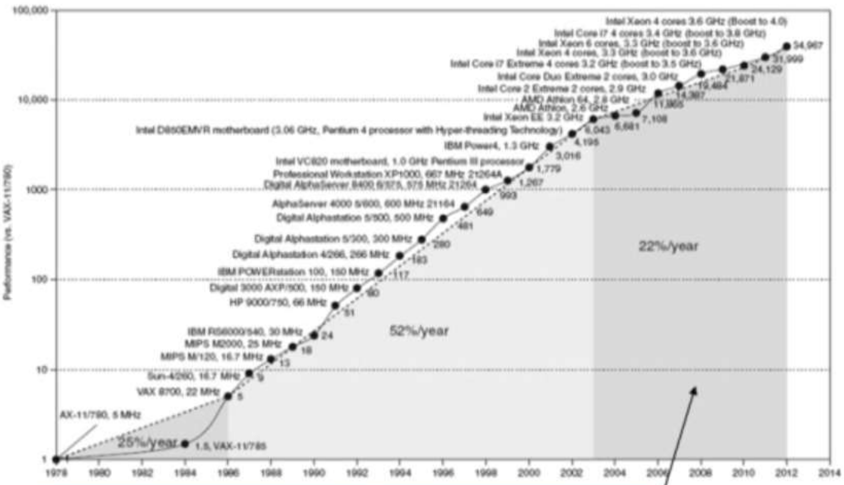


파워 = 커패시터의 용량 x 볼티지의용량 x 프리퀀시



- 열 문제때문에 클락 프리퀀시를 더이상 올릴 수 없다.
- 파워가 성능향상에 장애가 된다.
- 어떻게 성능을 향상 시킬것인가 고민.

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

- 멀티코어 : 하나의 칩에 여러개의 프로세스가 들어가는것. 기존에는 하나의 칩에 하나의 프로세스가 들어갔다.
- 멀티코어의 등장으로 여러개의 프로세스를 동시에 실행 시킬 수 있게 됐다.
- 멀티코어는 프로그래밍하기 어렵다. 디버깅도 아주 까다롭다
- 로드밸런싱의 문제 (ex. A는 일이 적고 B는 일이 많을때의 저울질 필요)
- 통신 및 동기화 최적화가 어렵다.

SPEC CPU Benchmark

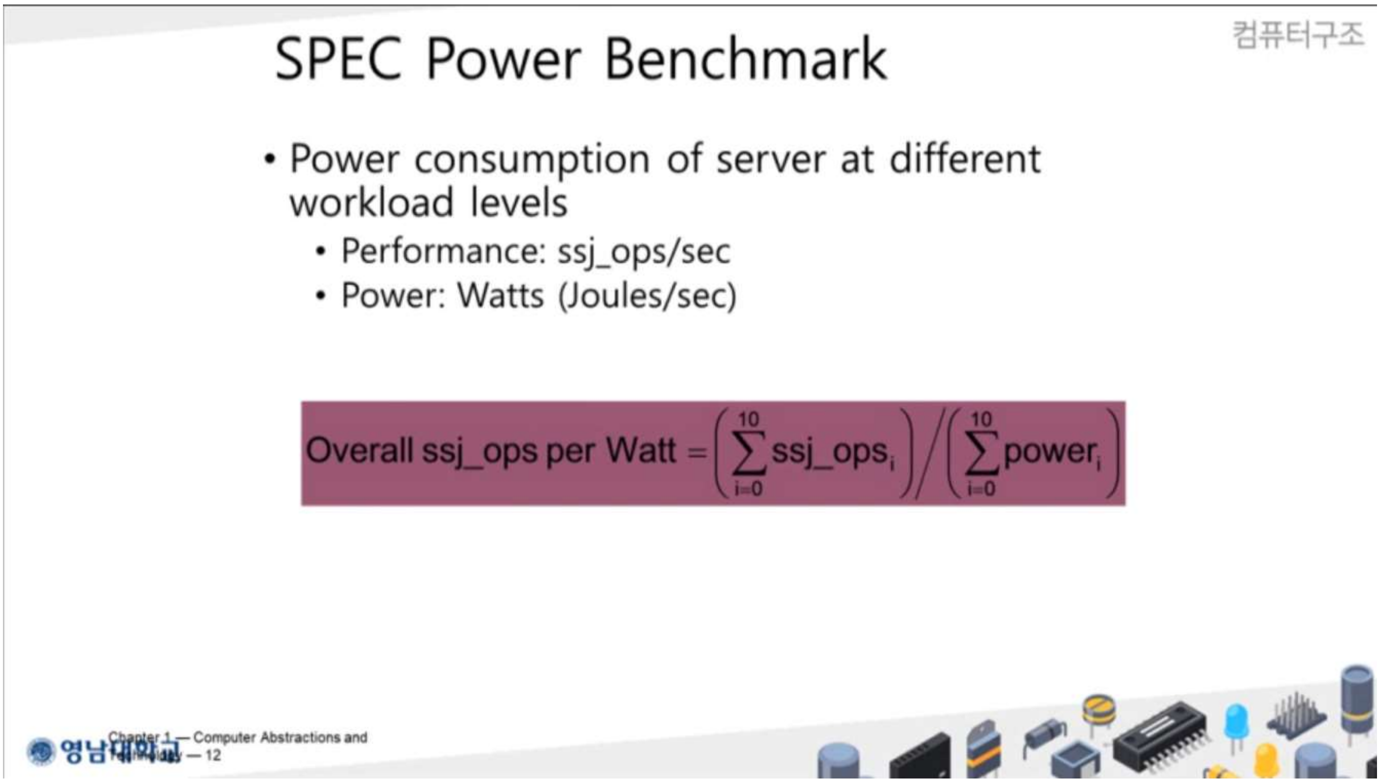
- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

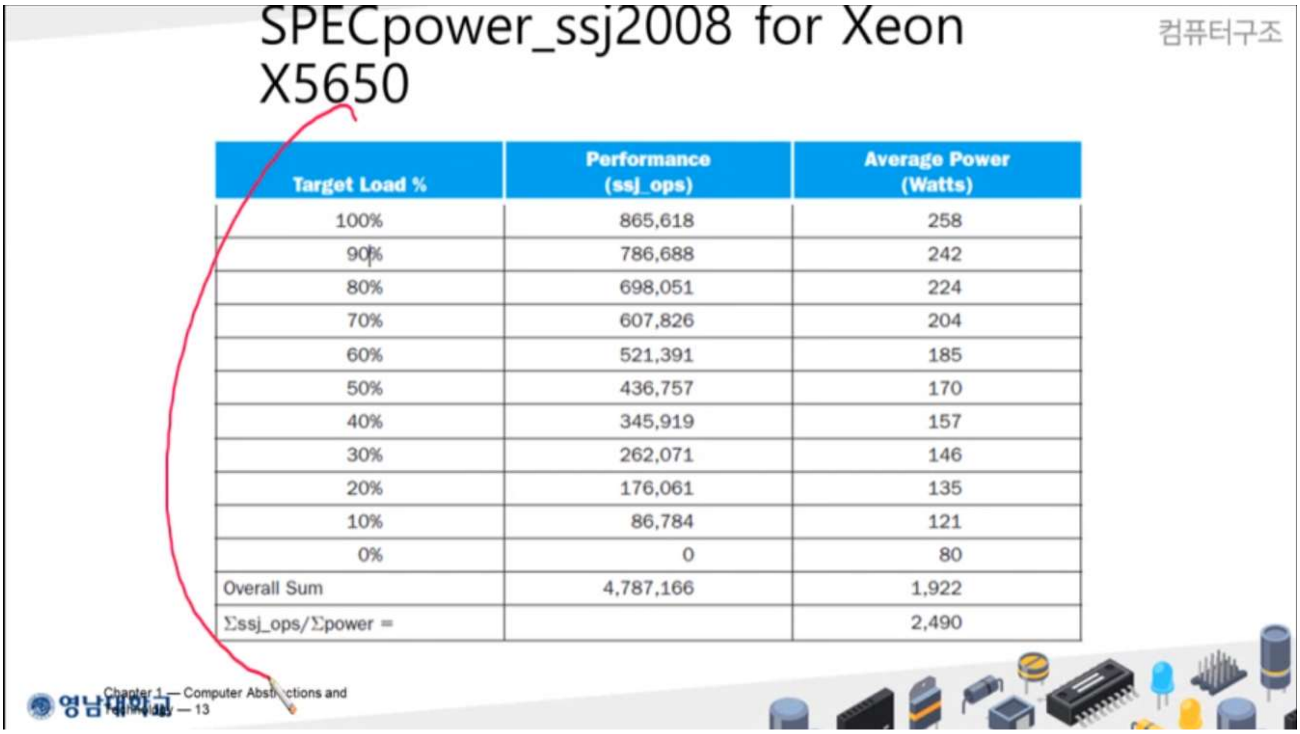
- SPEC은 Standard Performance Evaluation Corp의 약자이고, CPU, I/O, Web등등의 성능을 측정하는 표준 벤치마크이다.

CINT2006 for Intel Core i7 920

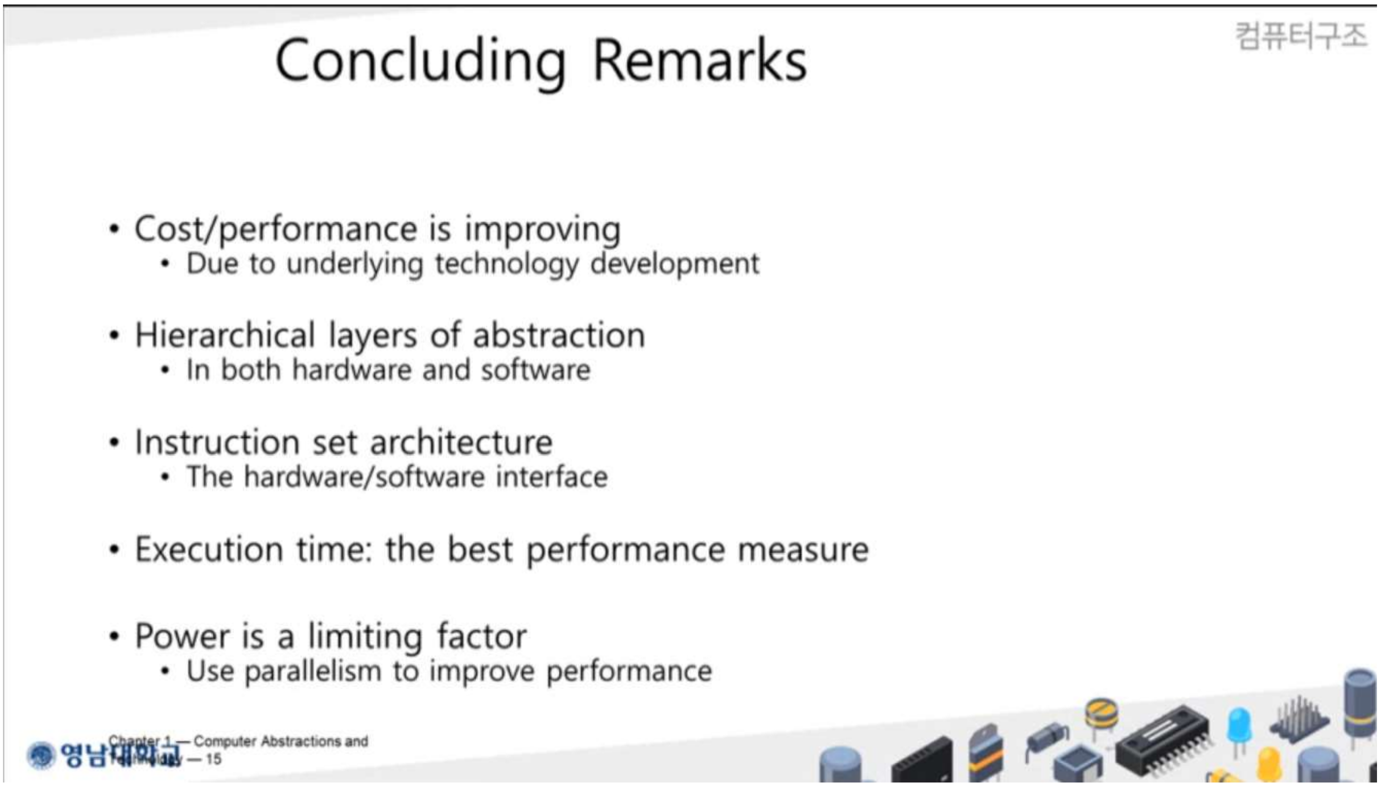
Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7



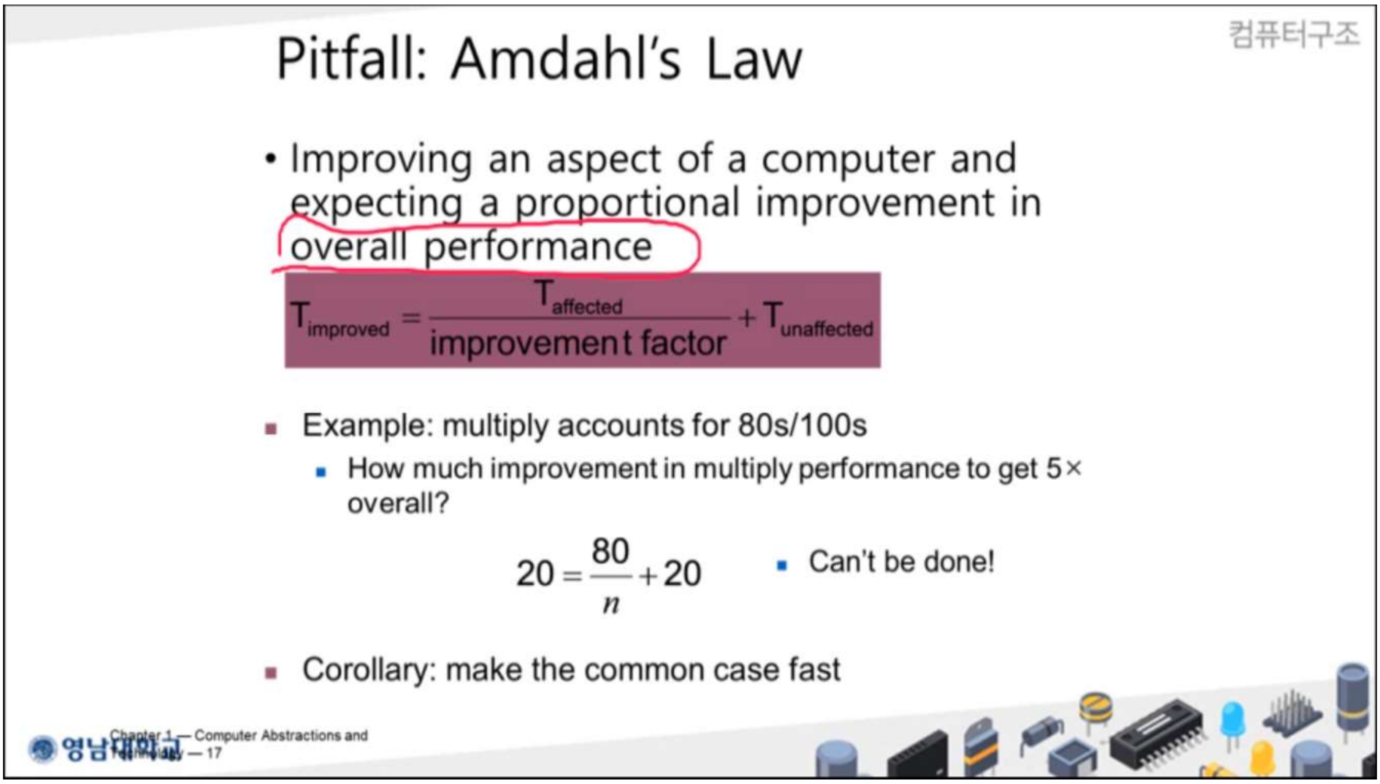
- 서버에서 파워 소비량을 측정하는 것.



Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490



Amdalhl's Law 암달의 법칙



- 어떤 시스템의 하나의 컴포넌트의 성능을 향상시키면 그 컴포넌트가 전체 시스템에서 차지하는 비율만큼 그 시스템의 성능이 향상된다 (당연한 이야기)

- 전체 성능에서 많은 포지션을 차지하는 부분의 성능을 향상시켜 만드는게 시스템에 더 영향을 많이 미친다. (진짜 당연한 이야기)

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

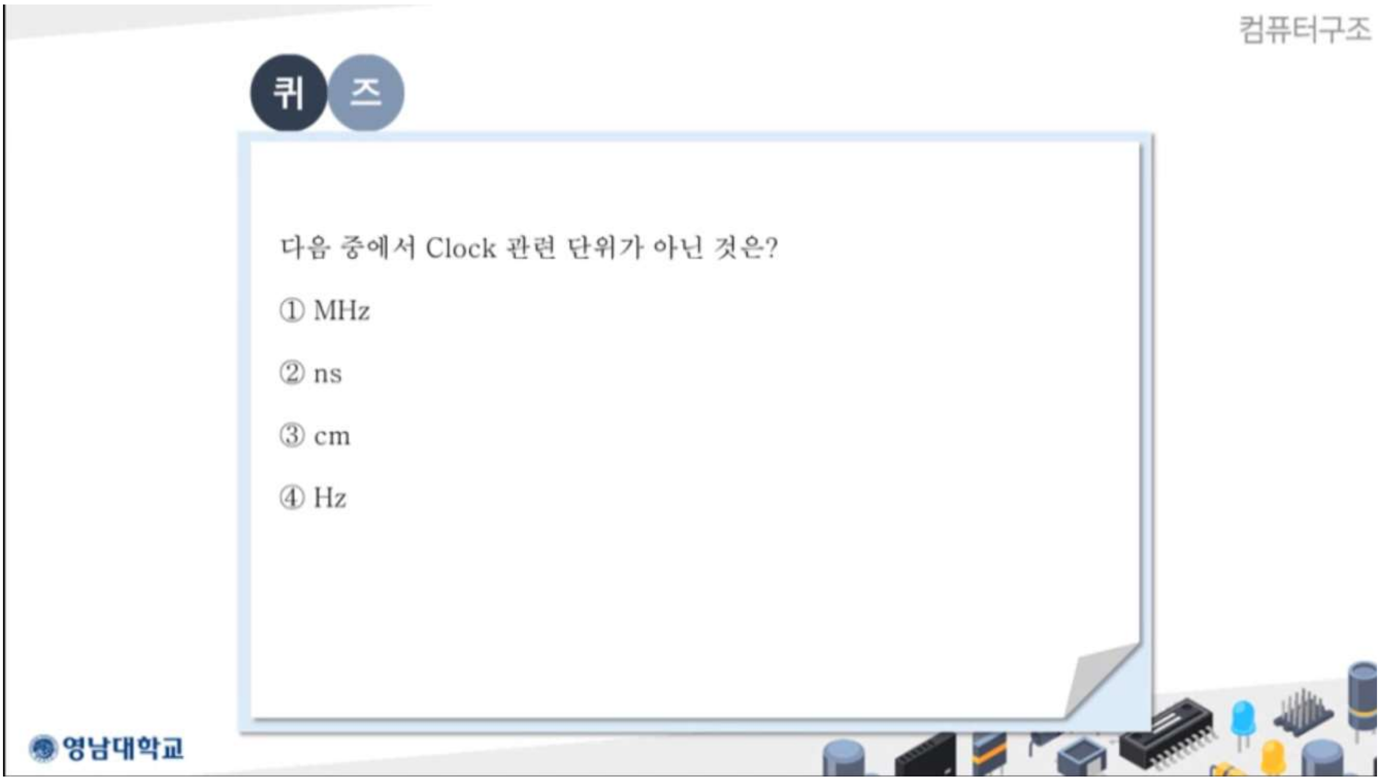
Pittall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

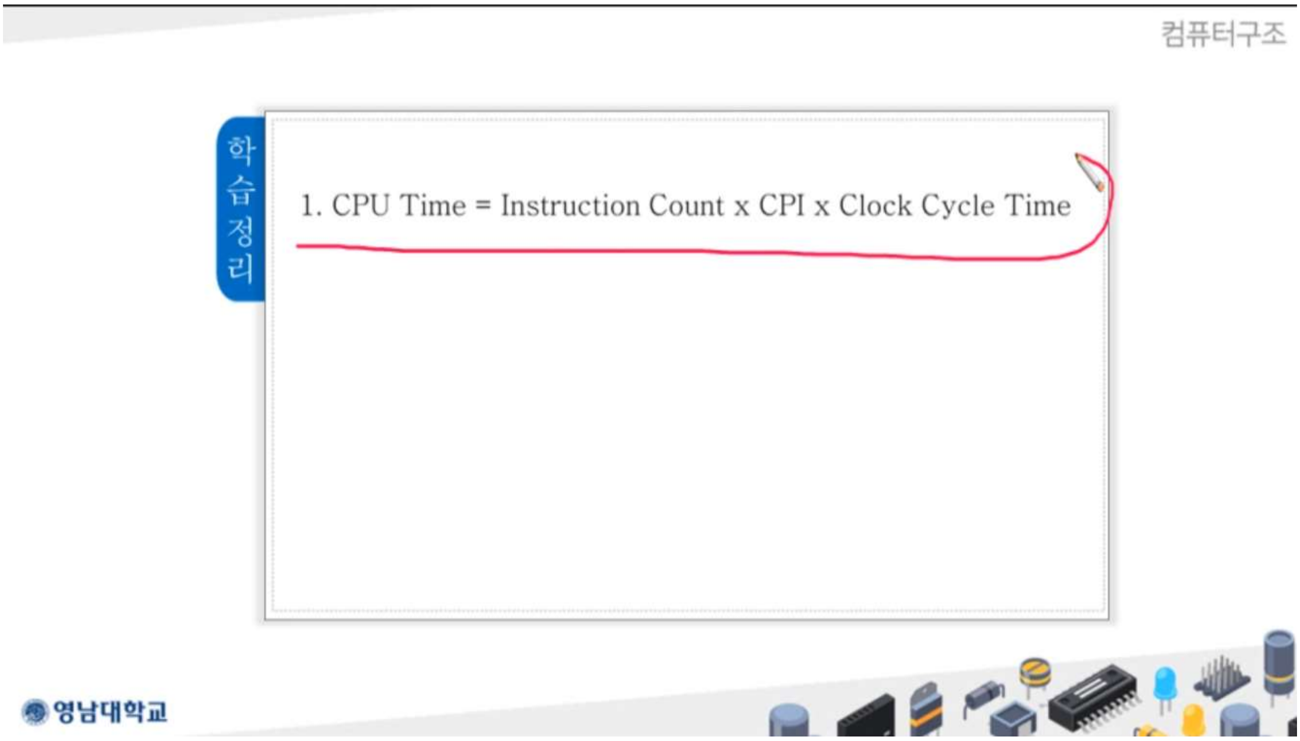
$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

- MIPS : 요즘에는 잘 안쓰이는 말, 성능지표,단점1 컴퓨터마다다른 Instruction Set Archietecture 고려하지않음, 단점2 CPI크기를 고려하지 않음
- CPI는 프로그램마다 달라질 수 있기에 MIPS는 올바른 성능 비교 지표가 아니다. 그래서 SPEC벤치마크와 같은것을 이용하는것이 올바른 방법이다.



- 답 3



도로로

필승

다음 포스트

Chapter3. Arithmetic for computers

0개의 댓글

댓글을 작성하세요

댓글 작성