Computer Architecture

컴퓨터구조 소개

Kyusik Chung
kchung@ssu.ac.kr

강의 안내 (1/6)

- 강의 웹사이트
  - 강의자료보기, 숙제보기, Q&A를 스마트캠퍼스 LMS를 통해 진행

- 강의 목표
  - 컴퓨터 내부 구조 및 동작원리 이해
  - 컴퓨터 하드웨어와 소프트웨어간의 상호작용 이해
  - 어셈블리 프로그래밍을 통한 컴퓨터 동작 이해

  성능최적화된 코딩에 도움이 되게 하자

  - 프로세서 성능 개선 기법 (Pipelining) 이해
  - 효율적인 메모리 설계 기법(메모리 계층 구조, 캐시, 가상 메모리) 이해

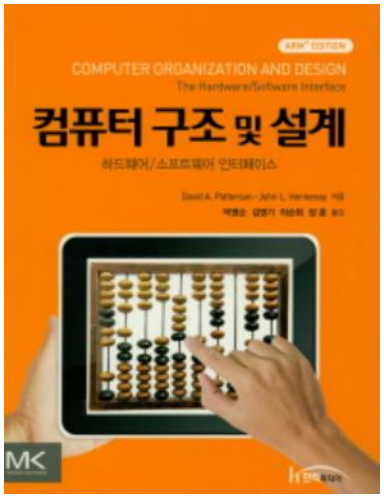
  고성능 HW 설계하는데 도움이 되게 하자

  OS 공부하는데 도움이 되게 하자

- 권장선수과목 – 디지털회로설계(2학년 2학기)
- 권장동반과목 – 시스템프로그래밍(2학년 2학기)
- 향후연계과목 – 운영체제(3학년 1학기)

강의 안내 (2/6)

- 교재 및 참고자료
  - 주교재: **D. A. Patterson & J. L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", ARM Ed., Elsevier, 2017**
    번역판: 박명순외 3인, "컴퓨터 구조 및 설계", **ARM ed.**, 한티미디어, **2018**
  - 참고교재: **Daniel W. Lewis, "ARM Assembly for Embedded Applications", 5th ed. 2019**
  - 참고교재: 안도히사, "프로세서를 지탱하는 기술", 제이펍, **2011**
- 수업 진행 방법
  - 이론(80%) 및 실습(20%) - 대면 및 사전녹화 동영상
  - 게시판에 질문올리면 응답하는 **Q&A**
  - 어셈블리 프로그래밍 실습
- 평가방법
  - 중간 평가 **35%**, 기말평가 **35%**, 과제 **25%**, 출석 **5%**
  - 대면통합시험: 중간 및 기말 시험 일정을 추후 결정



강의 안내 (3/6) - 교재내용1

교재내용대로 진행

Skip

1 Computer Abstractions and Technology 2

서론

1.1 Introduction 3
1.2 Eight Great Ideas in Computer Architecture 11
1.3 Below Your Program 13
1.4 Under the Covers 16
1.5 Technologies for Building Processors and Memory 24
1.6 Performance 28
1.7 The Power Wall 40
1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors 4
1.9 Real Stuff: Benchmarking the Intel Core i7 46
1.10 Fallacies and Pitfalls 49
1.11 Concluding Remarks 52
1.12 Historical Perspective and Further Reading 54
1.13 Exercises 54

2 Instructions: Language of the Computer 60

ARM 명령어 소개 – 보조자료로 대체하여 진행

2.1 Introduction 62
2.2 Operations of the Computer Hardware 63
2.3 Operands of the Computer Hardware 67
2.4 Signed and Unsigned Numbers 75
2.5 Representing Instructions in the Computer 82
2.6 Logical Operations 90
2.7 Instructions for Making Decisions 93
2.8 Supporting Procedures in Computer Hardware 100
2.9 Communicating with People 110
2.10 LEGv8 Addressing for Wide Immediates and Addresses 115
2.11 Parallelism and Instructions: Synchronization 125
2.12 Translating and Starting a Program 128
2.13 A C Sort Example to Put it All Together 137
2.14 Arrays versus Pointers 146

3 Arithmetic for Computers 186

컴퓨터에서의 연산 – 1,2장에서 부분적으로 다룬다

3.1 Introduction 188
3.2 Addition and Subtraction 188
3.3 Multiplication 191
3.4 Division 197
3.5 Floating Point 205
3.6 Parallelism and Computer Arithmetic: Subword Parallelism 230
3.7 Real Stuff: Streaming SIMD Extensions and Advanced Vector Extensions in x86 232
3.8 Real Stuff: The Rest of the ARMv8 Arithmetic Instructions 234
3.9 Going Faster: Subword Parallelism and Matrix Multiply 238
3.10 Fallacies and Pitfalls 242
3.11 Concluding Remarks 245
3.12 Historical Perspective and Further Reading 246
3.13 Exercises 249

4 The Processor 254

Pipeline 구조

4.1 Introduction 256
4.2 Logic Design Conventions 260
4.3 Building a Datapath 263
4.4 A Simple Implementation Scheme 271
4.5 An Overview of Pipelining 283
4.6 Pipelined Datapath and Control 297
4.7 Data Hazards: Forwarding versus Stalling 316
4.8 Control Hazards 328
4.9 Exceptions 336
4.10 Parallelism via Instructions 342
4.11 Real Stuff: The ARM Cortex-A53 and Intel Core i7 Pipelines 355
4.12 Going Faster: Instruction-Level Parallelism and Matrix Multiply 363
4.13 Advanced Topic: An Introduction to Digital Design Using a Hardware Design Language to Describe and Model a Pipeline and More Pipelining Illustrations 366

강의 안내 (4/6) - 교재내용2

Memory 계층구조

병렬처리

→ 교재내용대로 진행

강의 안내 (5/6)

- 강의내용
  - 컴퓨터 구조 소개 **(ch1-part1)** 및 디지털회로설계 기초 **(ch1-part2,** 보조자료**)**
  - **----------**
  - **ARM** 프로세서 구조**(ch2-part1)**
  - **ARM** 명령어 **(ch2-part2&3,** 보조자료**)**
  - **ARM** 어셈블리 프로그래밍 **(Lab0 ~ Lab4,** 보조자료**)**
  - **----------**
  - 프로세서 구조**(ch4-part1,** 파이프라이닝 개념**,** 명령어 파이프라이닝 학습**)**
  - 프로세서 구조**(ch4-part1, 5**단계 파이프라이닝 설계 및 동작학습**)**
  - 프로세서 구조**(ch4-part2, data hazard** 정의 및 해결방법 학습**)**
  - 프로세서 구조**(ch4-part2, control hazard** 정의 및 해결방법 학습**)**
  - **----------**
  - 메모리 구조 **(ch5-part1,** 메모리 계층구조 및 캐시의 동작 학습**)**
  - 메모리 구조**(ch5-part1,** 캐시 성능 측정 및 향상 방법 학습**)**
  - 메모리 구조**(ch5-part2,** 신용도있는 메모리 계층 구조 학습**)**
  - 메모리 구조**(ch5-part2,** 가상 메모리 개념**,** 주소변환 방법 학습**)**
  - **----------**
  - 병렬 프로세서**(ch6-part1,** 병렬처리 구조 및 병렬처리 프로그램개발학습**)**
  - 병렬 프로세서**(ch6-part2,** 클라우드 컴퓨팅 및 **IO** 동작원리 학습**)**

강의 안내 (6/6) -강의진도

주#	첫번째시간[사전녹화동영상]	교재내용	두번째시간[대면강의]	교재내용
1	강의1(컴퓨터구조소개)	ch1-part1	강의2(컴퓨터구조소개)	ch1-part2
2	강의3(ARM 프로세서구조)	ch2-part1	강의4(ARM 프로세서구조)	ch2-part1
3	강의5(ARM 프로그래밍)	ch2-part2	강의6(ARM 프로그래밍)	ch2-part2
4	강의7(ARM 프로그래밍예제)	ch2-part3	강의8(ARM 프로그래밍예제)	ch2-part3
5	Lab0(개발환경구축)		강의9(프로세서구조)	ch4-part1
6	Lab1(Factorial함수구현)		강의10(프로세서구조)	ch4-part1
7	Lab2(C pointer및stack overflow)		강의11(프로세서구조)	ch4-part1
8	Lab3(Bubble sorting구현)		중간고사	
9	Lab4(매트릭스곱셈구현)		강의12(프로세서구조)	ch4-part2
10	강의13(프로세서구조)	ch4-part2	강의14(메모리구조)	ch5-part1
11	강의15(메모리구조)	ch5-part1	강의16(메모리구조)	ch5-part1
12	강의17(메모리구조)	ch5-part2	강의18(메모리구조)	ch5-part2
13	강의19(메모리구조)	ch5-part2	강의20(병렬처리)	ch6-part1
14	강의21(병렬처리)	ch6-part2	강의22(병렬처리)	ch6-part2
15	강의23(최종정리)		기말고사	

파란색 사전녹화동영상

빨간색 대면강의

초록색 Lab(사전녹화동영상)

컴퓨터구조를 배워야 하는 이유 ? (1/2)

1. 성능 최적화된 SW 개발에 도움

- 프로그램이 하드웨어위에서 수행되는 과정을 배운다
  - 이 과정을 갖고 성능이 좋은 프로그램과 나쁜 프로그램을 구분할 수 있다.
  - 자료구조, 알고리즘의 중요성을 이해할 수 있다.
- vertical optimization (수직적 최적화) 가능
  - 응용프로그램관점에서만 바라보는 성능 최적화 (수평적 최적화) 대 하드웨어, OS 동작까지 고려한 응용프로그램 최적화(수직적 최적화)의 차이

2. 하드웨어 설계에 도움

- 컴퓨터에 대한 하드웨어에서 소프트웨어까지 계층적인 분류
- 하드웨어와 소프트웨어 사이 인터페이스
- 프로세서 수준에서의 성능 극대화 기법 학습 (파이프라이닝, 계층적인 메모리 구조)

컴퓨터구조를 배워야 하는 이유 ? (2/2)

3. OS 동작 이해에 도움 (3학년 1학기 OS 과목)

- 프로그램(process) context
  - 컴퓨터에서는 하나의 프로그램 수행하다가 다른 프로그램을 수행하게 한다. 수행하다만 프로그램의 상태를 저장했다가 나중에 복구해야 함.
  - 프로그램의 상태는 무엇을 의미하나 ? 레지스터의 값들, 메모리 값들로 구성 (컴구조에서 다루는 용어들)
- interrupt 처리과정
  - OS가 제때 중요한 역할을 할 수 있게 만들어 주는 게 interrupt mechanism.
  - interrrupt 발생하면 원래 처리해주어야 하는 일 + OS가 처리해야 하는 일을 수행
- 가상 메모리 구조 및 동작과정
  - 프로그래머들이 가상 메모리 환경에서 프로그래밍을 하게 해줌. 대신 OS에서 가상 메모리 운영 지원. 이에 대한 기초 지식 공부

2학년 2학기 학생들에게 전하고 싶은 내용(1/2)

1. 수강과목 선정시 고려사항

- 지금까지는 선택의 폭이 작음. 필수과목보다는 선택과목이 많지만 대부분 거의 비슷하게 수강신청 (선택과목 1-2개 차이)
- 3, 4학년으로 가면 선택에 따라 크게 차이가 남.

1) 과목들간 연관관계을 따져봐야 한다

- 과목간 연결성을 극대화하면 지식의 넓이와 폭이 커진다.
  (예. 프로그래밍 대 자료구조, 알고리즘 – 프로그래밍과 알고리즘간의 연관관계를 알면 프로그래밍에 대한 시야가 넓어진다)

2) 본인의 관심 방향 ?

3) 왜 그 과목을 들어야 하나 ?

2학년 2학기 학생들에게 전하고 싶은 내용(2/2)

2. 과목당 공부 시간 배정시 고려사항

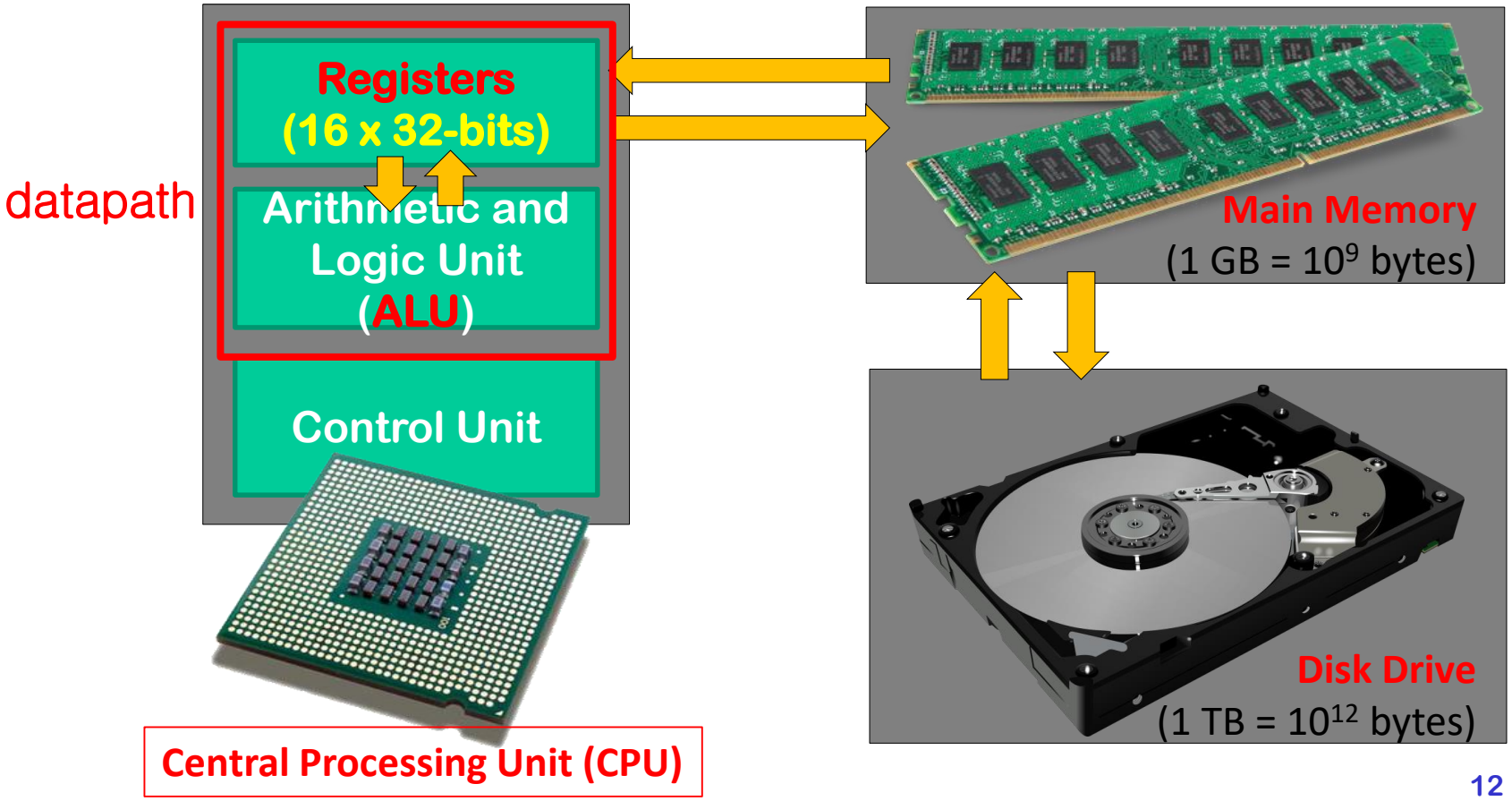
- 취업은 코딩 실력순
- 컴구조에 주당 1시간을 배정한다면 프로그래밍관련과목은 최소 3 시간 배정 (코딩 시간이 절대적으로 모자란다 !!!. 졸업시까지 코딩누적시간 2000 ~ 3000 시간 ?, 추가설명은 아래 자료 P18~26을 보세요 http://nclab.ssu.ac.kr/bbs2/board.php?bo_table=class_202101_k1_n&wr_id=31)

3. IT 기술 및 트랜드에 대한 이해 **(**전체를 보아야 한다**)**

- 지금까지 컴퓨터 기초, 특히 프로그래밍관련 과목들을 배웠슴.
- IT 기술 및 트랜드(전체)에 대한 이해도가 낮음.
- 컴구조 과목에서 2번의 숙제를 통해 IT 기술 및 트랜드를 공부할 예정

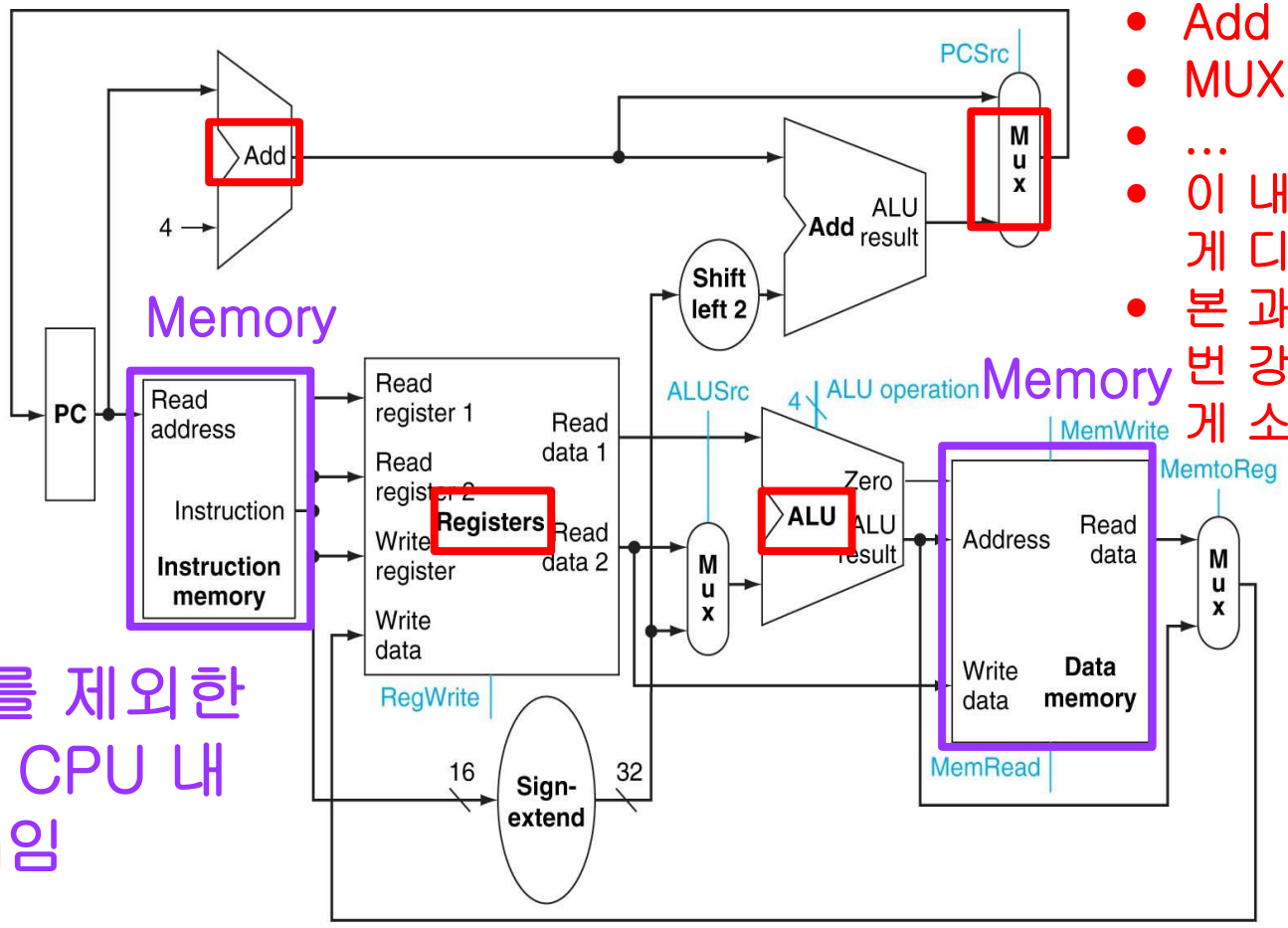
컴퓨터 내부 구조 및 동작원리 이해(1/3)

- **Hardware 시스템측면**
  - 컴퓨터의 내부구조(CPU, Memory, I/O) 및 각 구성요소 세부구조
  - 각 구성요소의 동작원리



컴퓨터 내부 구조 및 동작원리 이해(2/3)

- Simple Datapath for MIPS Architecture



Memory를 제외한 나머지는 CPU 내 datapath임

- ALU 동작방법 ?
- Register 동작방법?
- Add 동작방법 ?
- MUX 동작방법 ?
- …
- 이 내용을 공부하는 게 디지털회로설계임
- 본 과목에서는 다음 번 강의에서 간단하게 소개함

컴퓨터 내부 구조 및 동작원리 이해(3/3)

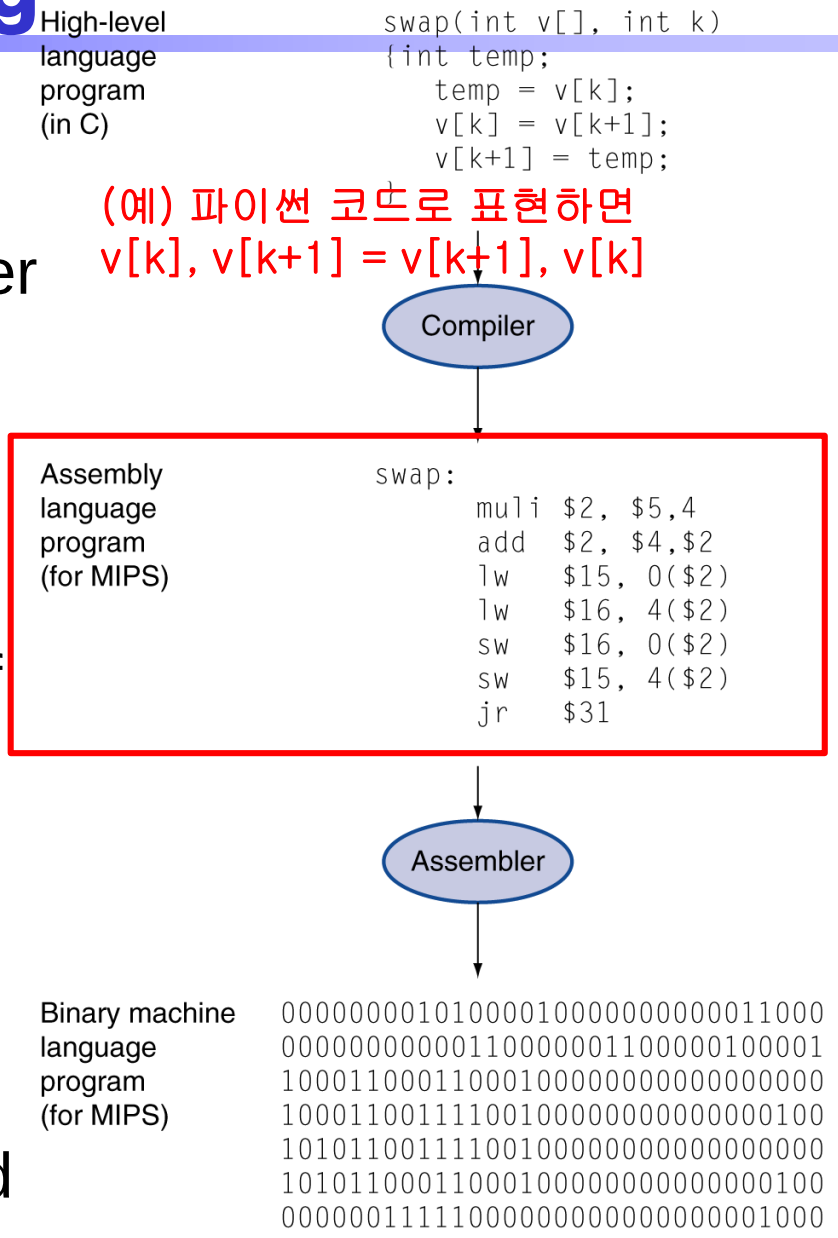
- **Software 시스템측면: 컴퓨터는 하나의 instruction set으로 정의할 수 있는 machine**
  - **Instruction ?**
    - 컴퓨터가 해독하여 수행할 수 있도록 설계된 명령어
    - 2진수로 표현됨
    - (예) 두개의 숫자를 갖고 덧셈하고 결과저장하라는 명령어
  - **instruction과 컴퓨터에서 수행되는 일과의 관계**
    - 무엇이 컴퓨터를 제어하는가 ? ---- CPU
    - 무엇이 CPU를 제어하는가 ? ---- Control Unit (CU)
    - 무엇이 CU를 제어하는가 ? ---- instruction
    - 무엇이 instruction을 제어하는가 ? - programmer가 만든 program에 의해 (Program = an ordered set of instructions)

컴퓨터 하드웨어와 소프트웨어간의 상호작용 이해

- (ex) C로 작성한 Z=X+Y 코드가 컴파일되면 다음과 같은 몇 개의 기계어(instruction, 명령어)들로 바뀐다.
  - LDR R1, M[X];변수 X가 저장된 메모리값, M[X],을 R1으로 복사
  - LDR R2, M[Y]; 변수 Y가 저장된 메모리값, M[Y],을 R2로 복사
  - ADD R3, R1, R2; 레지스터 R1과 R2를 더하여 결과를 R3에 저장
  - STR M[Z], R3; R3값을 변수 Z가 저장된 메모리에 저장
- 이 명령어들이 컴퓨터내부에서 어느 구성요소들에 의해 어떻게 수행되는지를 (sequential circuit 분석하는 것과 같이 timing에 근거하여) 이해한다.
  - Sequential circuit ? – 디지털회로설계 과목 학습 내용. 다음시간에 소개

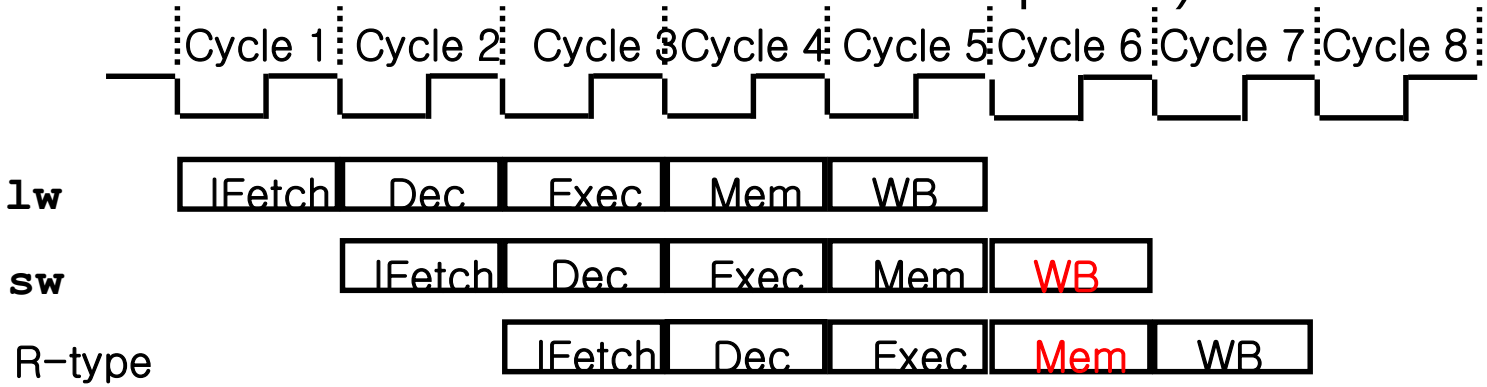
Assembly Programming

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation (machine code)**
  - Binary digits (bits)
  - Encoded instructions and data



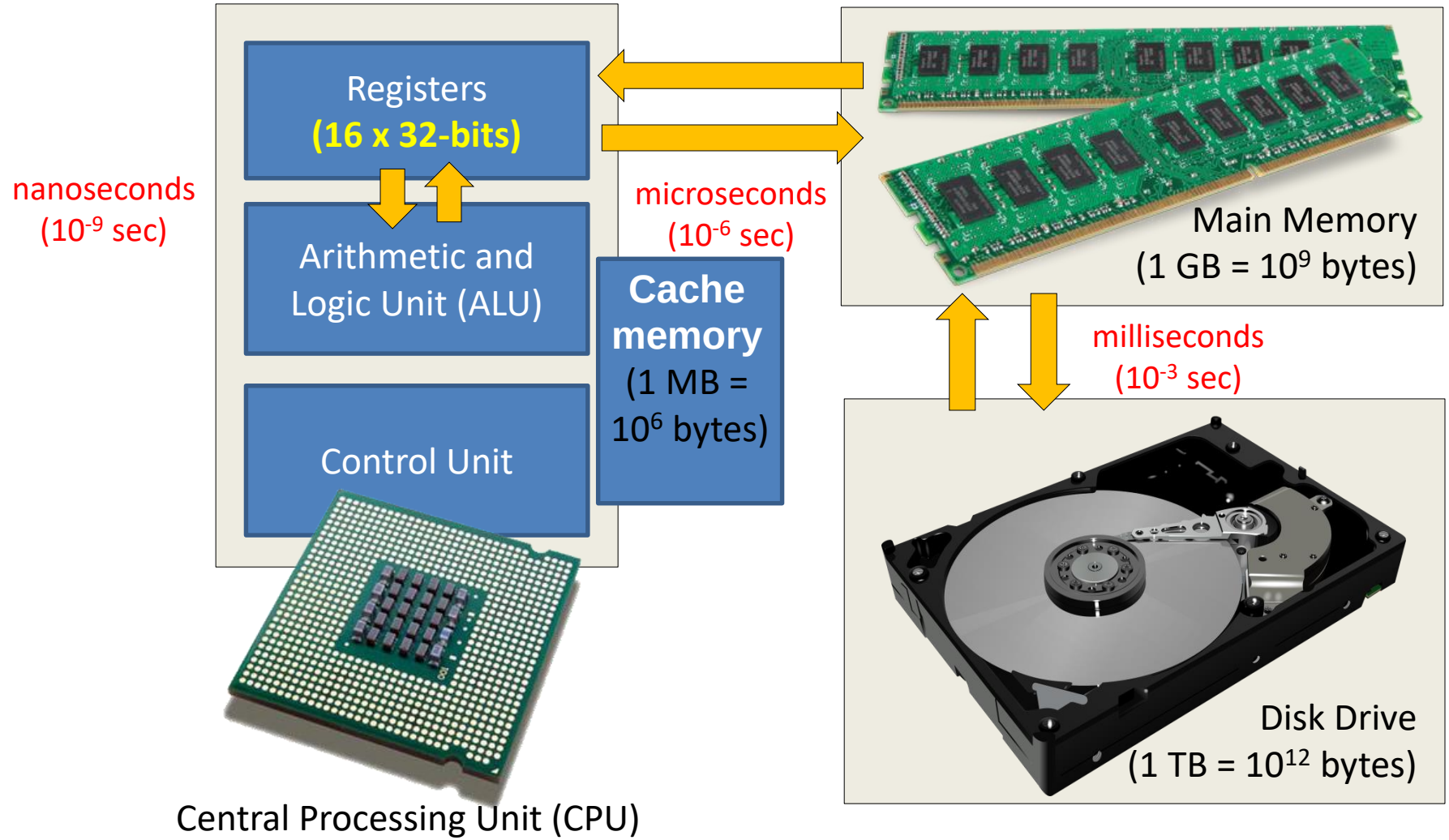
프로세서 성능 개선 기법 **(Pipelining)** 예

- **Start the next instruction before the current one has completed**
  - improves throughput - total amount of work done in a given time
  - instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is *not* reduced



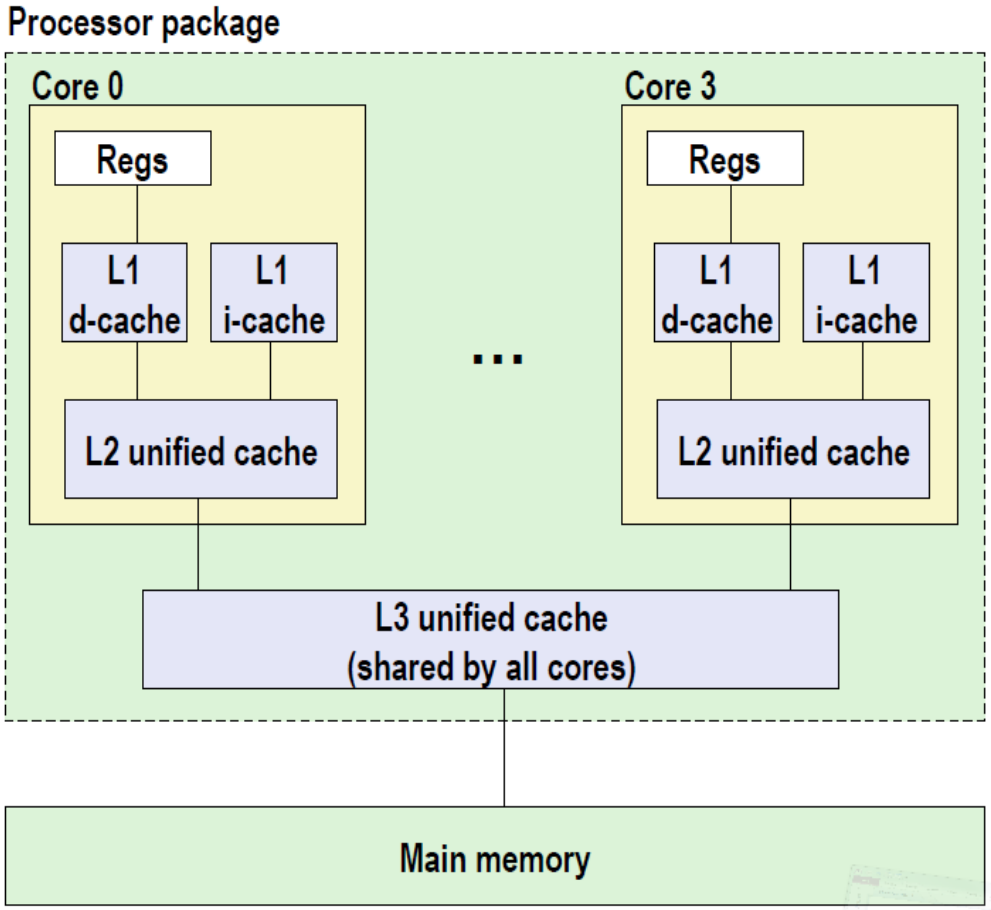
- **clock cycle (pipeline stage time) is limited by the slowest stage**
- **for some stages don't need the whole clock cycle (e.g., WB)**
- **for some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction**

COMPUTER COMPONENTS



효율적인 메모리 설계 기법(메모리 계층 구조, 캐시) 예

Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

Computer Abstraction and Technology

[Adapted from the lecture notes of Prof. Byung-gi Kim]

Contents

Computer Architecture

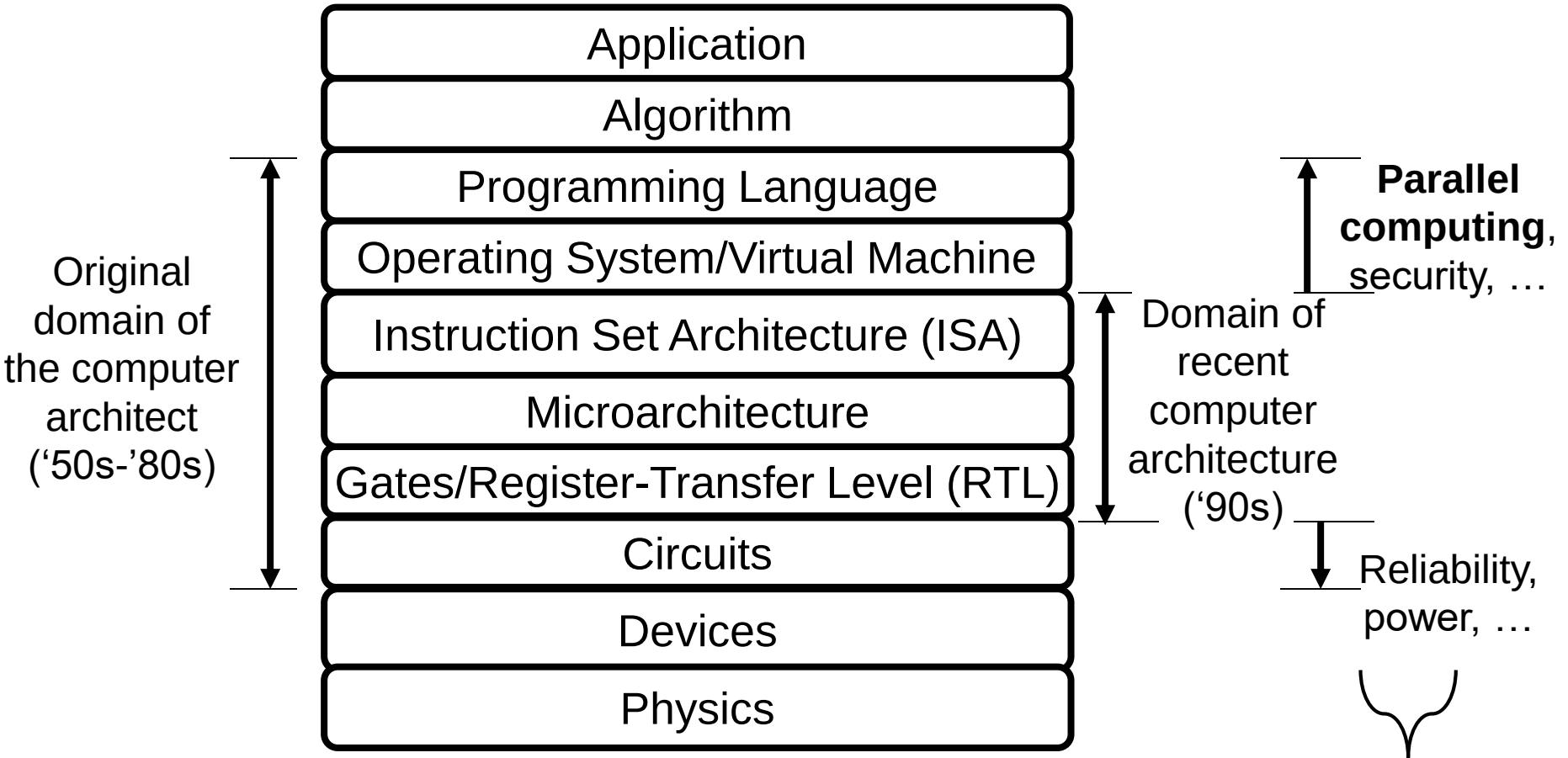
- = Instruction Set Architecture (ISA)
- Those aspects of the instruction set available to programmers, independent of the hardware on which the instruction set was implemented
- The term computer architecture was first used in 1964 by Gene Amdahl, G. Anne Blaauw and Frederick Brooks, Jr., the designers of IBM System/360
- IBM System/360 was a family of computers
  - all with the same architecture
  - but with a variety of organization (=implementations).

Architecture vs. Implementation

- **Instruction set architecture (=Architecture)**
  - Interface between hardware and lowest-level software
  - Defines what a computer system does in response to a program and a set of data
  - Includes all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory access, I/O, ...
  - (예) MIPS, IA-32, IA-64, Sparc, PowerPC, IBM 390, etc.
- **Implementation (=Organization=Microarchitecture)**
  - Hardware that obeys the architecture abstraction
  - Defines how a computer does in response to a program and a set of data
  - (예) 8086, 386, 486, Pentium, Pentium II, Pentium 4, etc.

Abstraction Layers in Modern Systems

Abstraction(추상화) & Refinement(구체화) – software 개발방법론



Abstraction is concerned with hiding lower levels of detail; it moves from lower to higher levels. Refinement is the movement from higher levels of detail to lower levels. Both concepts are necessary in developing software.

1.1 Introduction – Classes of Computers

- **Personal computers**
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- **Server computers**
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- **Supercomputers**
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- **Embedded computers**
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

1.1 Introduction – The PostPC Era(1/2)

Worldwide Device Shipments by Device Type, 2016-2019 (Millions of Units)

Device Type	2016	2017	2018	2019
Traditional PCs (Desk-Based and Notebook)	220	204	193	187
Ultramobiles (Premium)	50	59	70	80
Total PC Market	270	262	264	267
Ultramobiles (Basic and Utility)	169	160	159	156
Computing Device Market	439	423	423	423
Mobile Phones	1,893	1,855	1,903	1,924
Total Device Market	2,332	2,278	2,326	2,347

Source: Gartner (January 2018)

1.1 Introduction – The PostPC Era(2/2)

- **Personal Mobile Device (PMD)**
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- **Cloud computing**
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

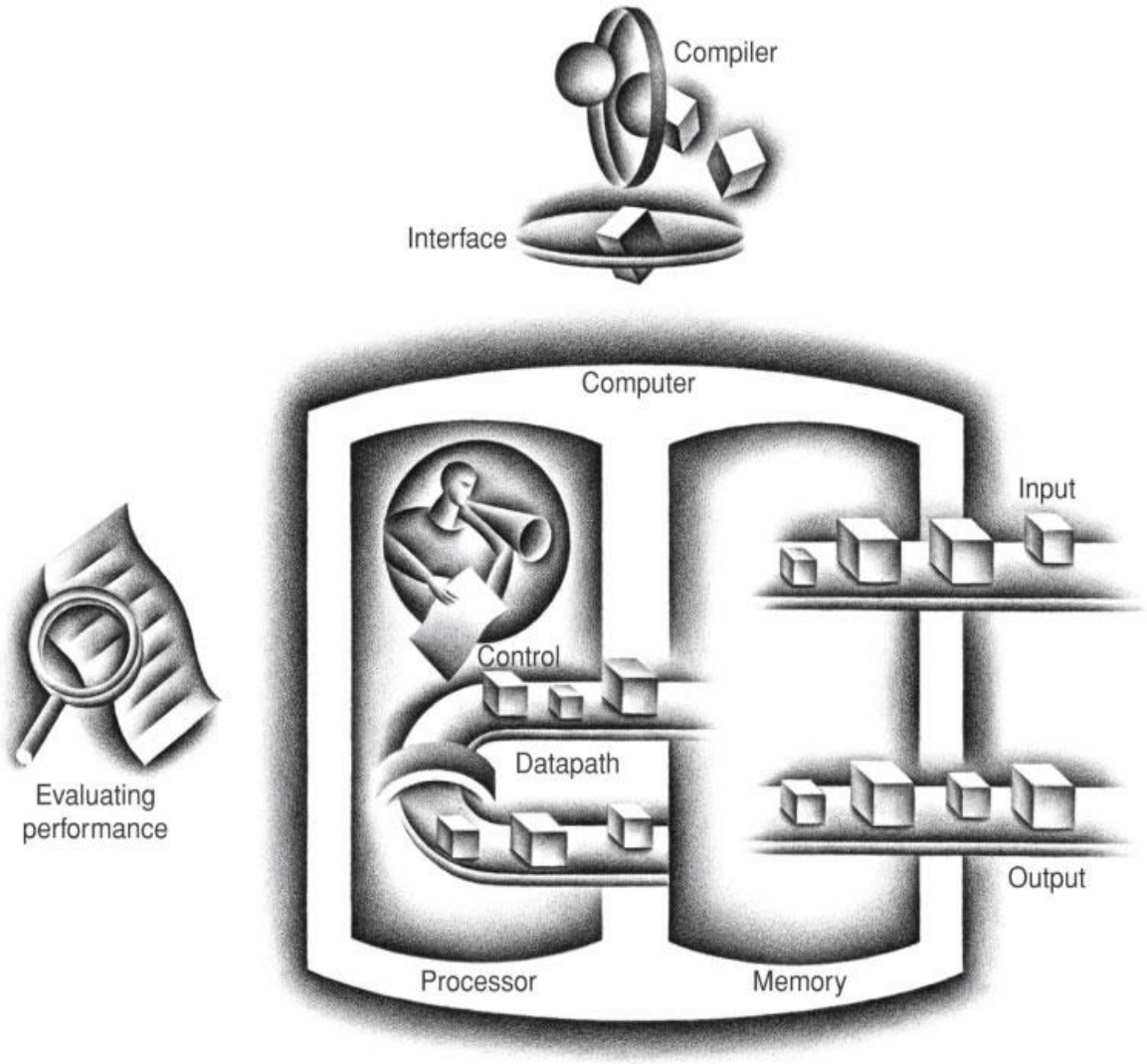
1.1 Introduction – what to learn in this book

1. How are program written in a high-level language translated into the language of the hardware, and how does the hardware execute the resulting program?
2. What is the interface between the soft ware and the hardware, and how does soft ware instruct the hardware to perform needed functions?
3. What determines the performance of a program, and how can a programmer improve the performance?
4. What techniques can be used by hardware designers to improve performance?
5. What techniques can be used to improve energy efficiency?
6. What are the reasons for and the consequences of the recent switch from sequential processing to parallel processing?
7. What great ideas did computer architects come up with that lay the foundation of modern computing?

What to affect the Program Performance?

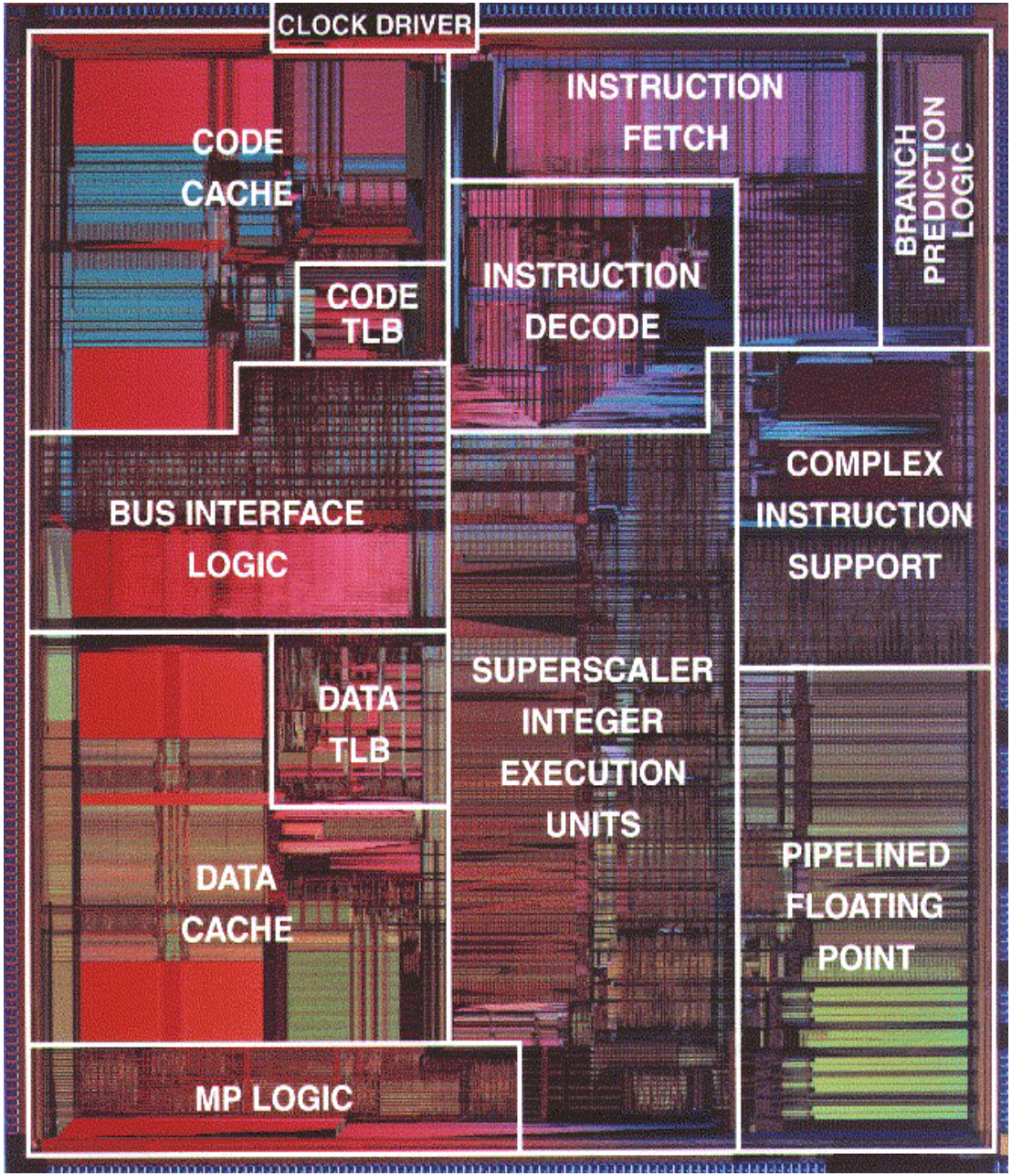
- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, architecture**
  - Determine number of machine instructions executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system (including OS)**
  - Determines how fast I/O operations are executed

Five Classic Components



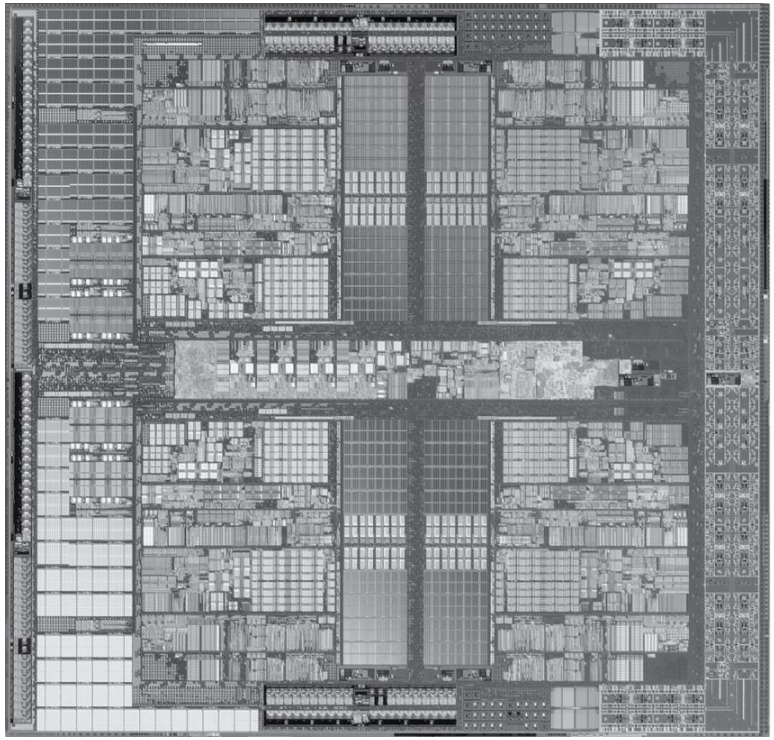
- **CPU**
  - Datapath: performs operations on data
  - Control: sequences the operations of datapath, memory, input, and output
- **Memory**
- **IO**
  - Input
  - Output

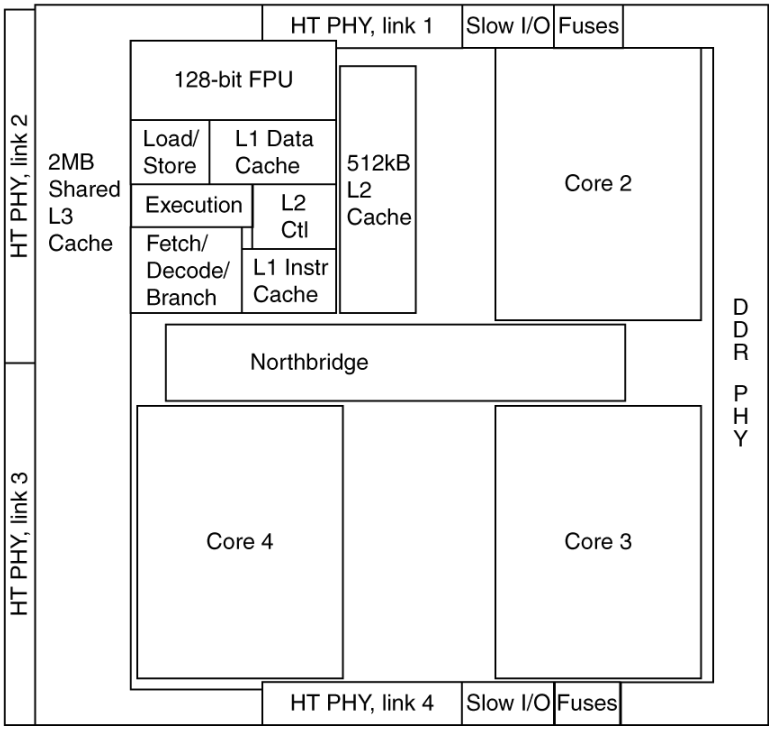
Intel Pentium



AMD Barcelona Processor

- **4 processors or "cores"**





Apple A5 Chip in iPad 2

FIGURE 1.9 The processor integrated circuit inside the A5 package. The size of chip is 12.1 by 10.1 mm, and it was manufactured originally in a 45-nm process (see Section 1.5). It has two identical ARM processors or cores in the middle left of the chip and a PowerVR graphical processor unit (GPU) with four datapaths in the upper left quadrant. To the left and bottom side of the ARM cores are interfaces to main memory (DRAM). (Courtesy Chipworks, www.chipworks.com)

2^x vs. 10^y

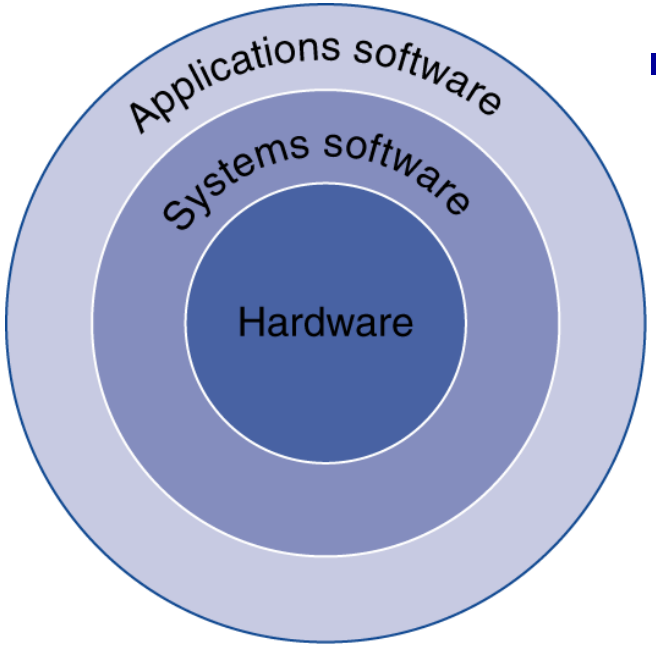
Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte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

FIGURE 1.1 The 2^X vs. 10^Y bytes ambiguity was resolved by adding a binary notation for all the common size terms. In the last column we note how much larger the binary term is than its corresponding decimal term, which is compounded as we head down the chart. These prefixes work for bits as well as bytes, so *gigabit* (Gb) is 10^9 bits while *gibibits* (Gib) is 2^{30} bits.

1.2 8 Great Ideas in Computer Architecture

1. Design for Moore's Law
2. Use Abstraction to Simplify Design
3. Make the Common Case Fast
4. Performance via Parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierarchy of Memories
8. Dependability (fault-tolerance) via Redundancy

1.3 Below Your Program(1/2)



- **Application software**
  - Written in high-level language
- **System software**
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- **Hardware**
  - Processor, memory, I/O controllers

1.3 Below Your Program(2/2) – Levels of Program Code

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

Assembler

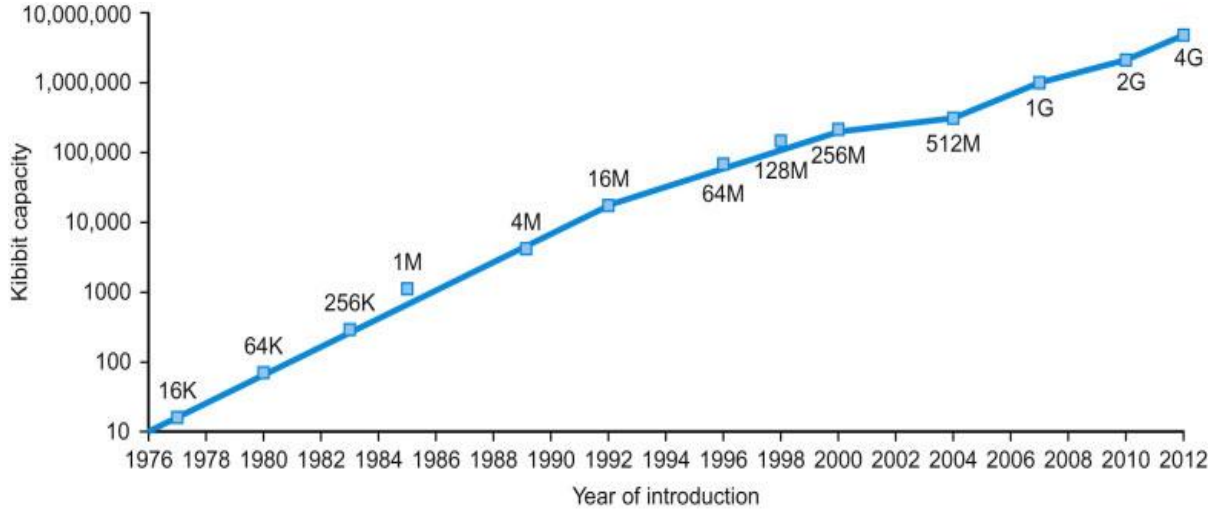
Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

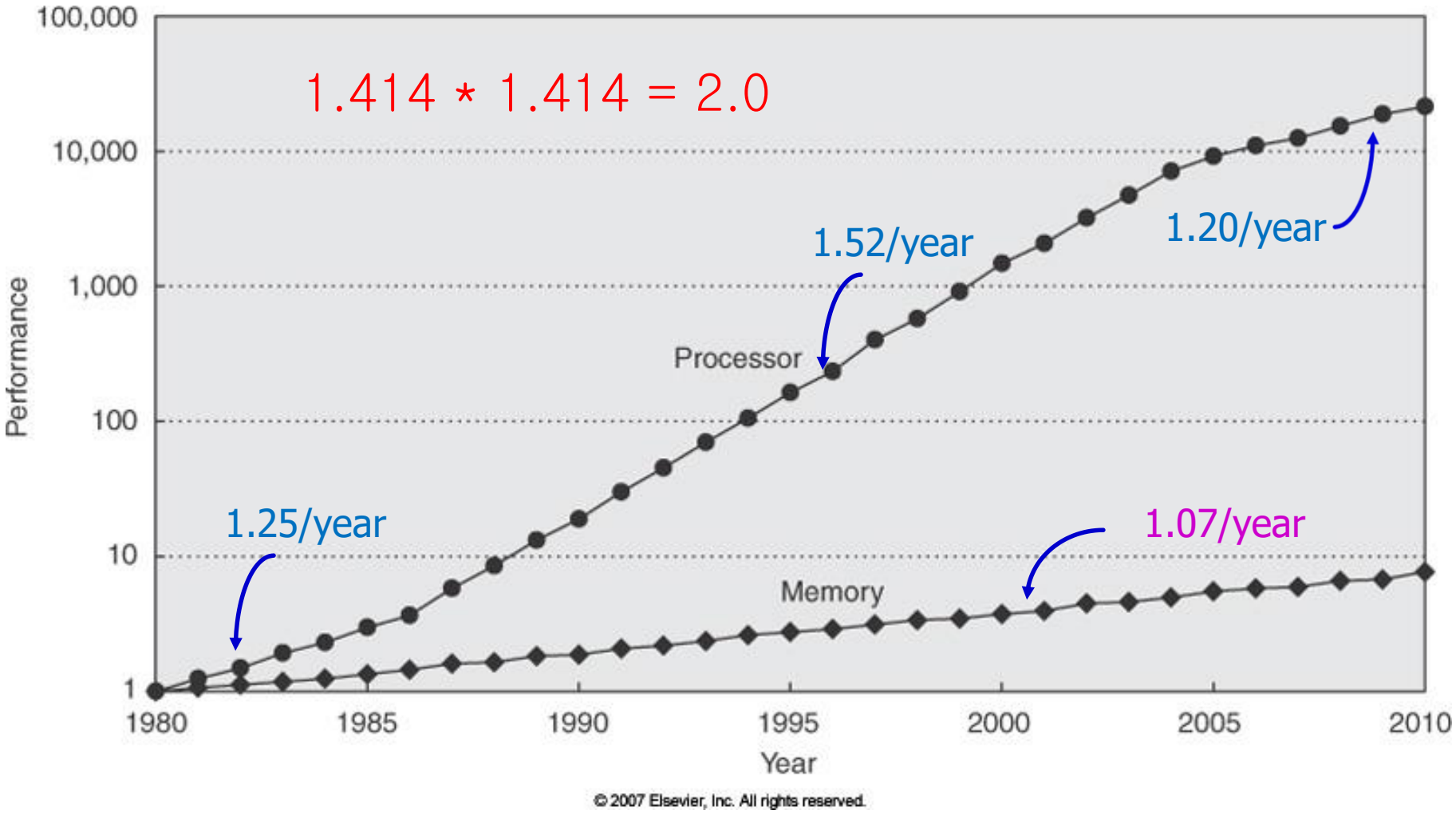
1.6 Performance

Year	Technology	Relative performance/unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	IC	900
1995	VLSI	2,400,000
2013	ULSI	250,000,000,000

- **Moore's law** : 트랜지스터 집적도가 2년에 2배로 증가

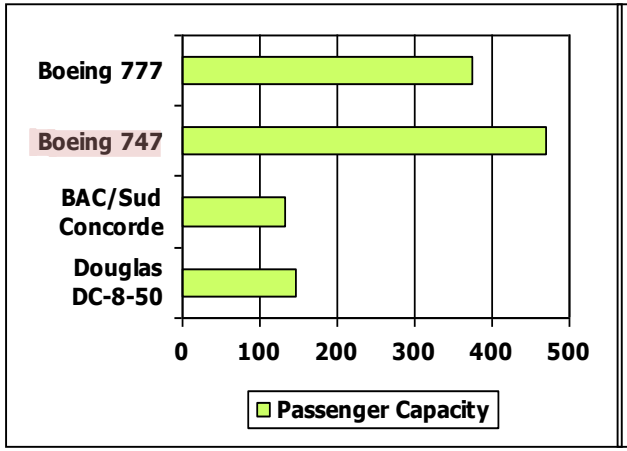


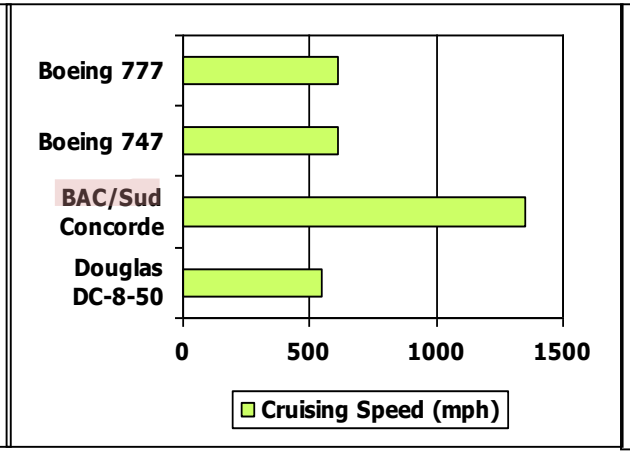
Processor-Memory Performance Gap

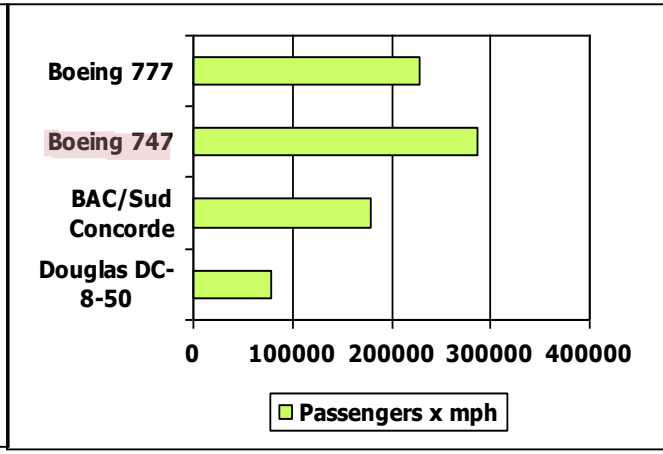


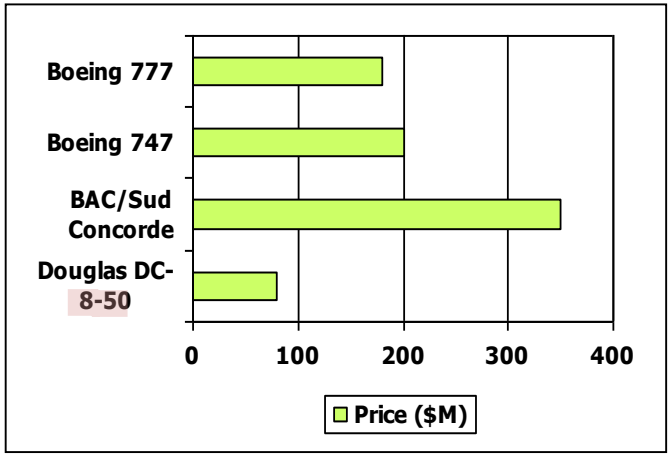
Defining Performance

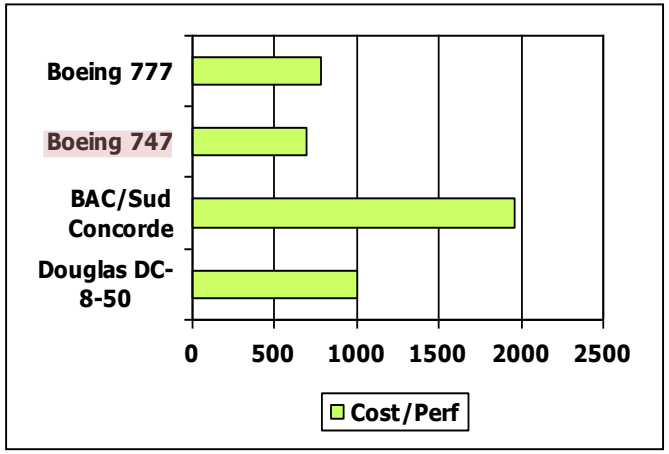
- **Which airplane is the best ?**











Performance of a Computer

- **Response time (= execution time)**
  - The time between the start and completion of a task
- **Throughput (= bandwidth)**
  - The number of tasks completed per unit time
- **Performance and execution time**
  - $\text{Performance}_x = 1 / \text{Execution time}_x$
- **X is n times faster than Y**

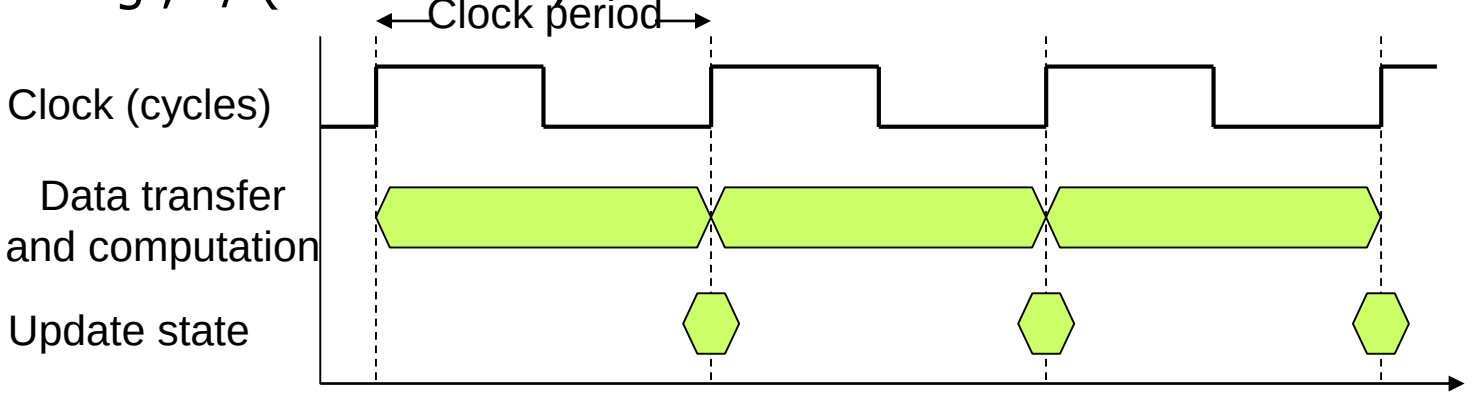
$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Exceution Time}_X} = n$$

Measuring Performance

- **Definitions of time**
  - Wall-clock time = Response time = Elapsed time
    - Total time to complete a task
    - Including disk accesses, memory accesses, I/O activities, OS overhead and etc.
    - Determines system performance
  - CPU execution time = CPU time
    - The time CPU spends computing for this task
    - Not including time spent waiting for I/O or running other programs
    - CPU time = User CPU time + System CPU time
    - Determines CPU performance
- **Definitions of performance**
  - System performance: based on elapsed time
  - CPU performance: based on user CPU time
- **We will focus on CPU performance for now!**

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock
- **Clock period=clock cycle time: duration of a clock cycle**
  - e.g., 250ps = 250×10^{-12}s= 0.25ns = 0.25×10^{-9}s
- **Clock frequency (rate): cycles per second**
  - e.g., 1/ (0.25×10^{-9}s) = 4.0×10^{9}Hz = 4000MHz = 4.0GHz



CPU Performance and Its Factors

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- **Performance improved by**
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

Example: Improving Performance

- 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?**

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

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

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

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

Instruction Performance

Clock Cycles = IC x CPI

- **Instruction Count (IC) for a program**
  - Determined by program, ISA and compiler
- **Average Cycles Per Instruction (CPI)**
  - Determined by CPU hardware
  - If different instructions have different CPI,
    - average CPI affected by instruction mix

Example: Using the Performance Equation

- Same instruction set architecture, same program
- Clock cycle time$_A$ = 250ps, CPI$_A$ = 2.0
- Clock cycle time$_B$ = 500ps, CPI$_B$ = 1.2
- **Which is faster, and by how much ?**

[Answer]

- Let I = instruction count for the program.
- CPU time$_A$ = IC$_A$ x CPI$_A$ x clock cycle time$_A$
  = I x 2.0 x 250 ps = 500 x I ps
- CPU time$_B$ = I x 1.2 x 500 ps = 600 x I ps
- Then

$$\frac{\text{CPU Performance}_A}{\text{CPU Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times \text{I ps}}{500 \times \text{I ps}} = 1.2$$

- Thus, A is 1.2 times faster than B for this program.

The Classic CPU Performance Equation

CPU Time = IC x CPI x clock cycle time

= (IC x CPI) / clock rate

- **Example: Comparing Code Segments**
  - **Which code sequence executes the most instructions?**
  - **Which will be faster ?**
  - **What is the CPI for each sequence ?**

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

[Answer]

- instruction count$_1$ = 2 + 1 + 2 = 5 and
  instruction count$_2$ = 4 + 1 + 1 = 6
  Thus (1) executes fewer instructions.
- CPU clock cycles$_1$ = 2x1 + 1x2 + 2x3 = 10 and
  CPU clock cycles$_2$ = 4x1 + 1x2 + 1x3 = 9
  Thus (2) is faster.
- CPI$_1$ = CPU clock cycles$_1$ / instruction count$_1$
  = 10 / 5 =2.0
  CPI$_2$ = 9 / 6 = 1.5
  (2) has lower CPI.

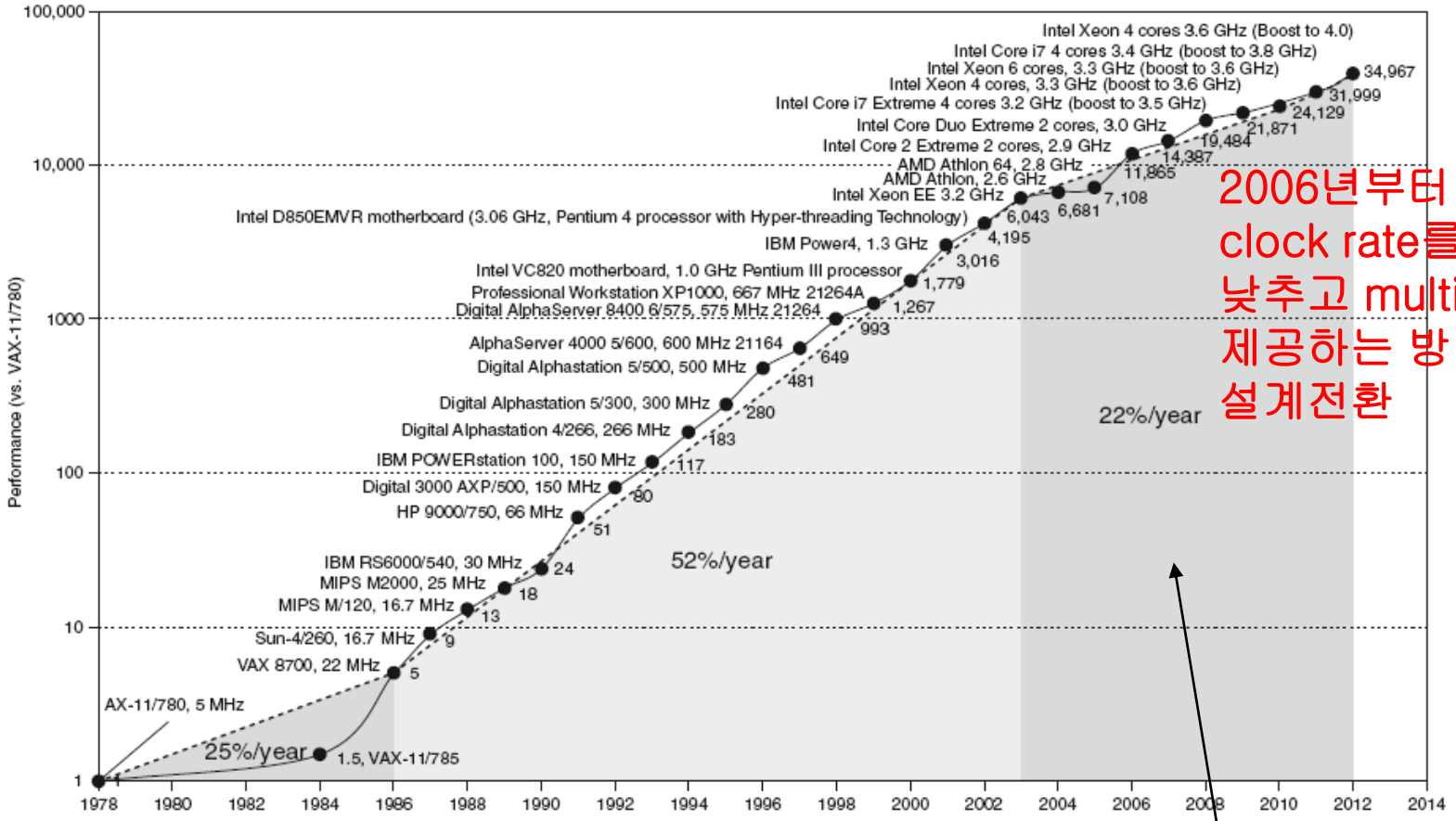
The BIG Picture

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

IC CPI T_C

- **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

1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors(1/2)



2006년부터 CPU clock rate를 낮추고 multicore 제공하는 방법으로 설계전환

Constrained by power, instruction-level parallelism, memory latency

1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors(2/2)

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

1.11 Concluding Remarks

- **Cost/performance is improving**
  - Due to underlying technology development
- **Hierarchical layers of abstraction**
  - In both hardware and software
- **Instruction set architecture**
  - The hardware/software interface
- **Execution time**

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

  - The best performance measure
- **Power is a limiting factor**
  - Use parallelism to improve performance