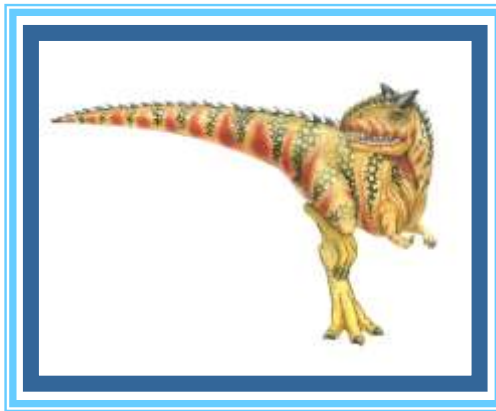


Chapter 1: Introduction

Review of CPU Organization





Class Information

❑ Textbook:

○ Operating System : Concepts 9th Ed. Or Later – By A. Silberschatz, *et al.*

○ Other Books will be referred as necessary

➤ 컴퓨터구성 & 컴퓨터 구조

➤ You must understand Ch.5 of 컴퓨터구성 before next class



❑ Class Slide:

○ e-class

○ Email: klee@dongguk.edu : Send me Questions and Suggestions

○ Office: 신공학관 10123 (Phone: 2260-3843)

○ Office Hours: Thu. 15:00 pm. (by appointment, only)

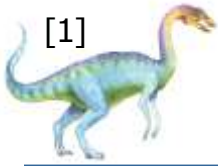
❑ Grades:

○ Exams (30% + 40% = 70% or 30% + 20% + 20% = 70%)

○ Homework (20%)

➤ Exercise Problems, Surveys or Essays

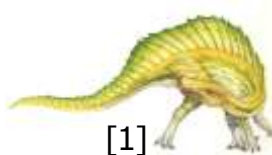




[1]

Introductory Questions

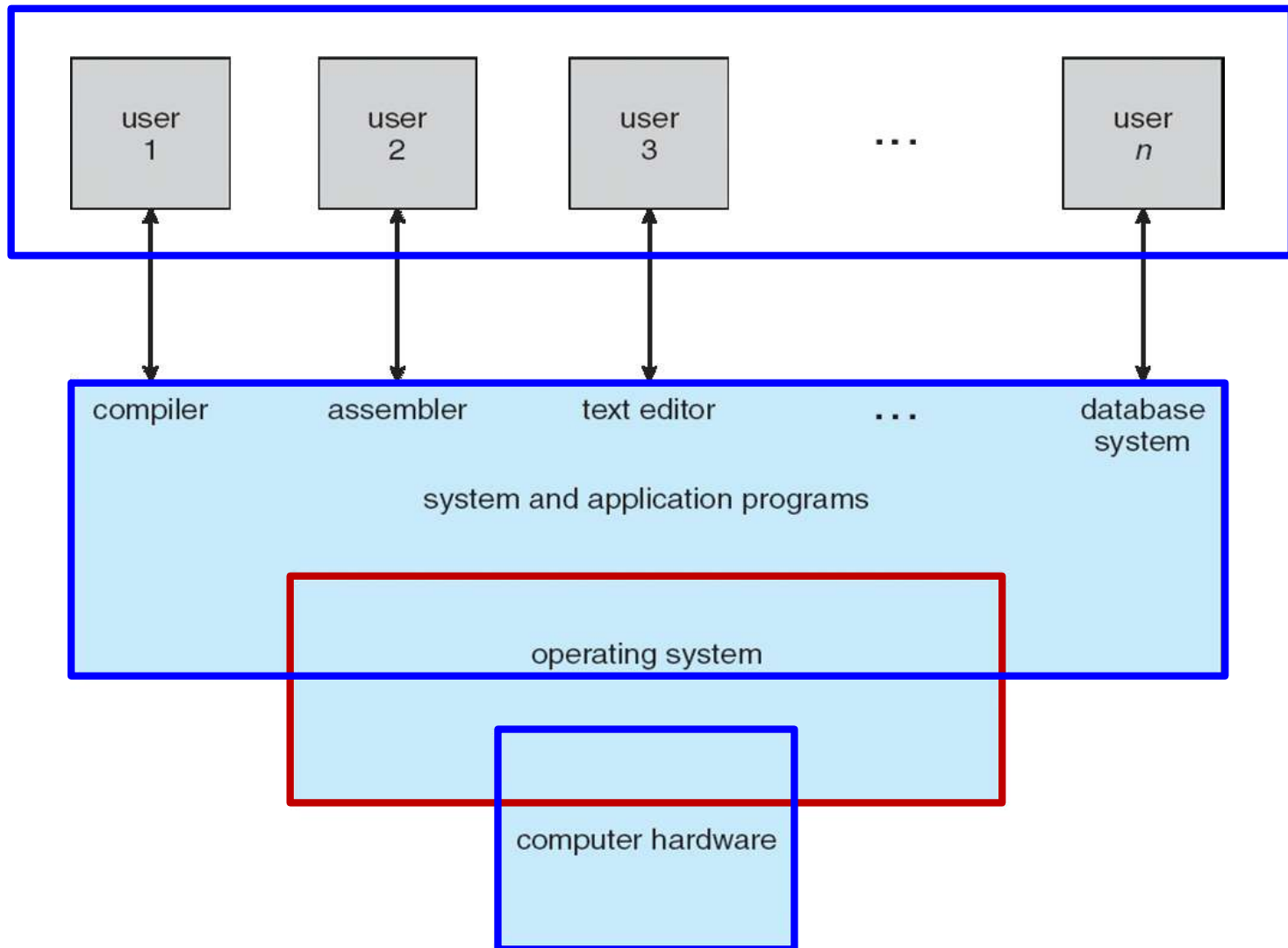
- What can you do **with computers without Operating Systems**?
- **What** does Operating Systems do?
- What are the **purposes** of the OS in doing those tasks?
- **How** the OS **performs the tasks** to accomplish its **purposes** (algorithms)?
- Can you **implement** those algorithms? Then, **analyze**? Then, **design**?
- **How much the hardware are engaged?**



[1]



Four Components of a Computer System



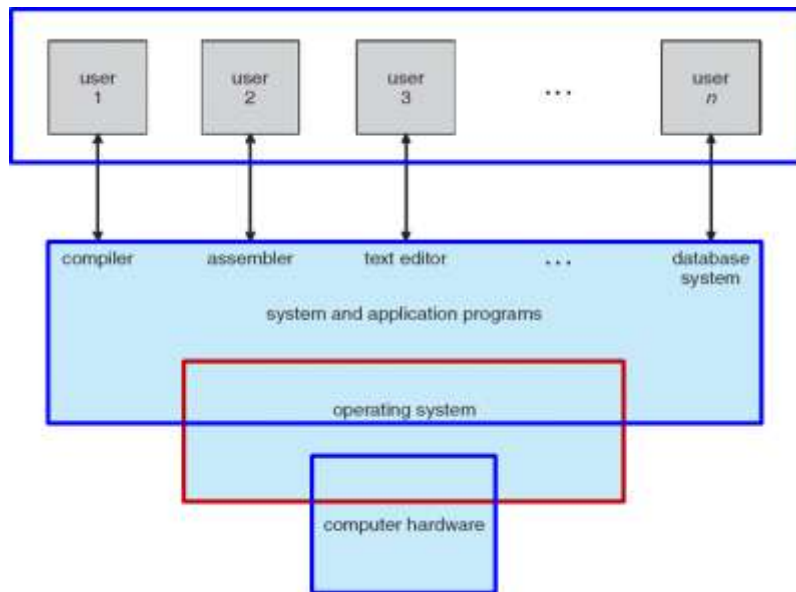
[1]





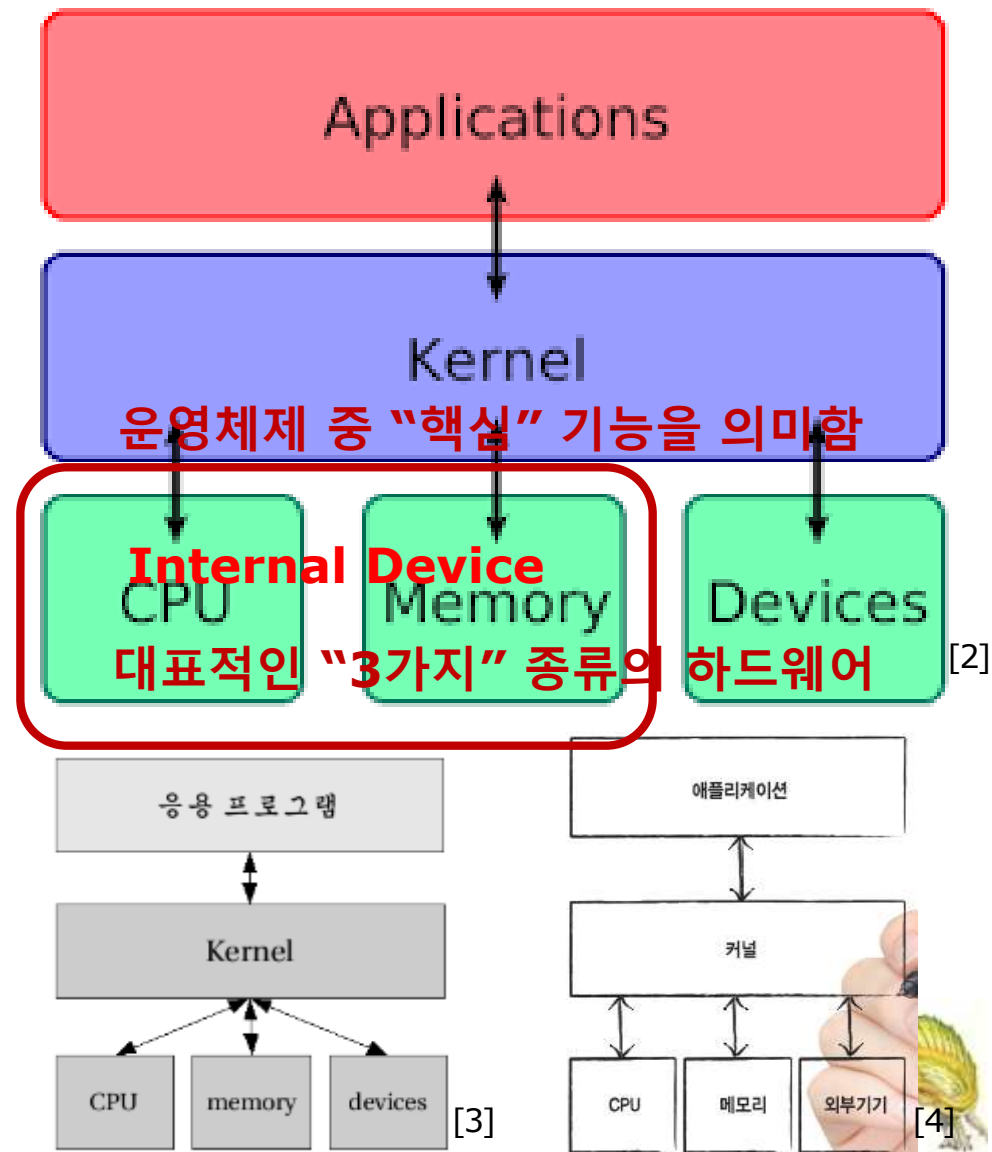
Four Components of a Computer System

More figures on the Web



Device
= IO Device
= **External Device**
= Peripheral
= IO Peripheral
= External Peripheral

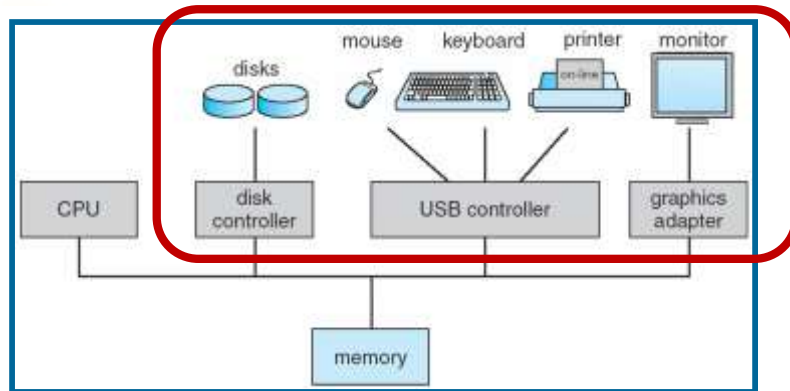
1.5





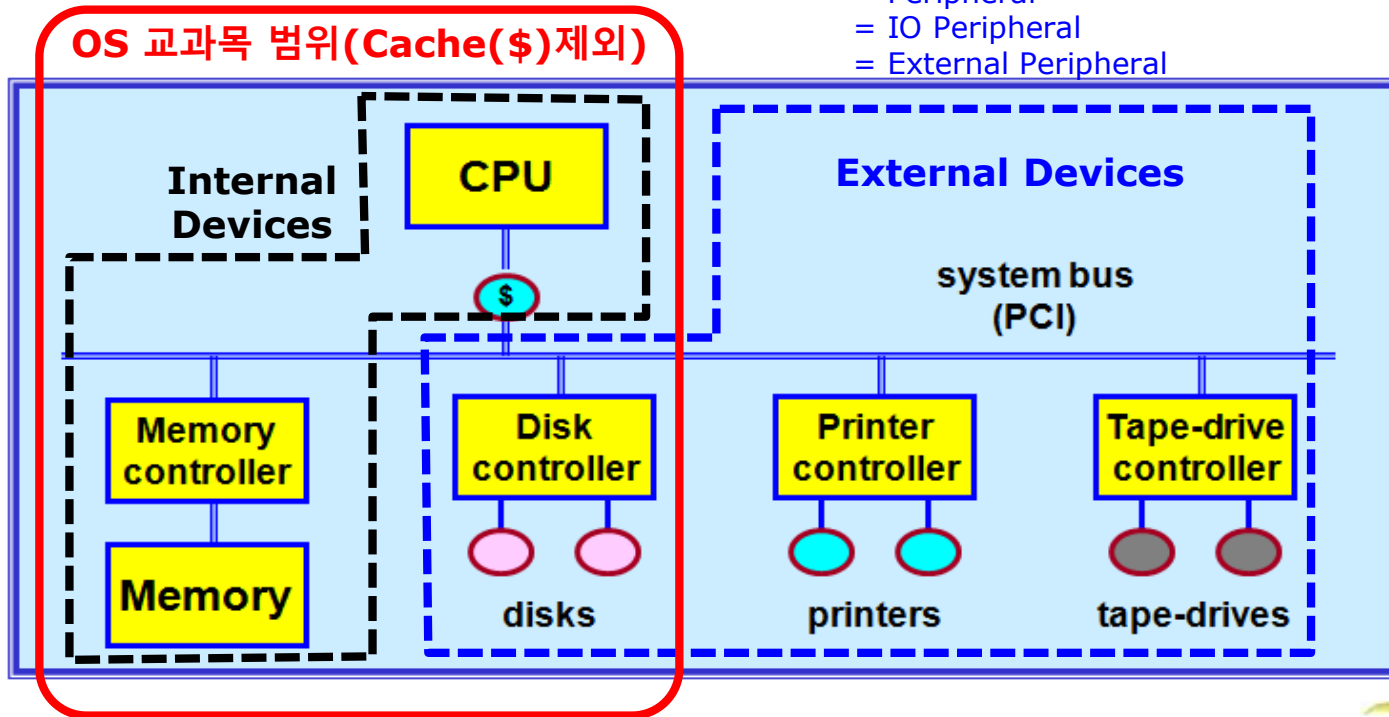
Understanding Computer System Architecture

[1]



Device
= IO Device
= **External Device**
= Peripheral
= IO Peripheral
= External Peripheral

OS 교과목 범위(Cache(\$\$)제외)



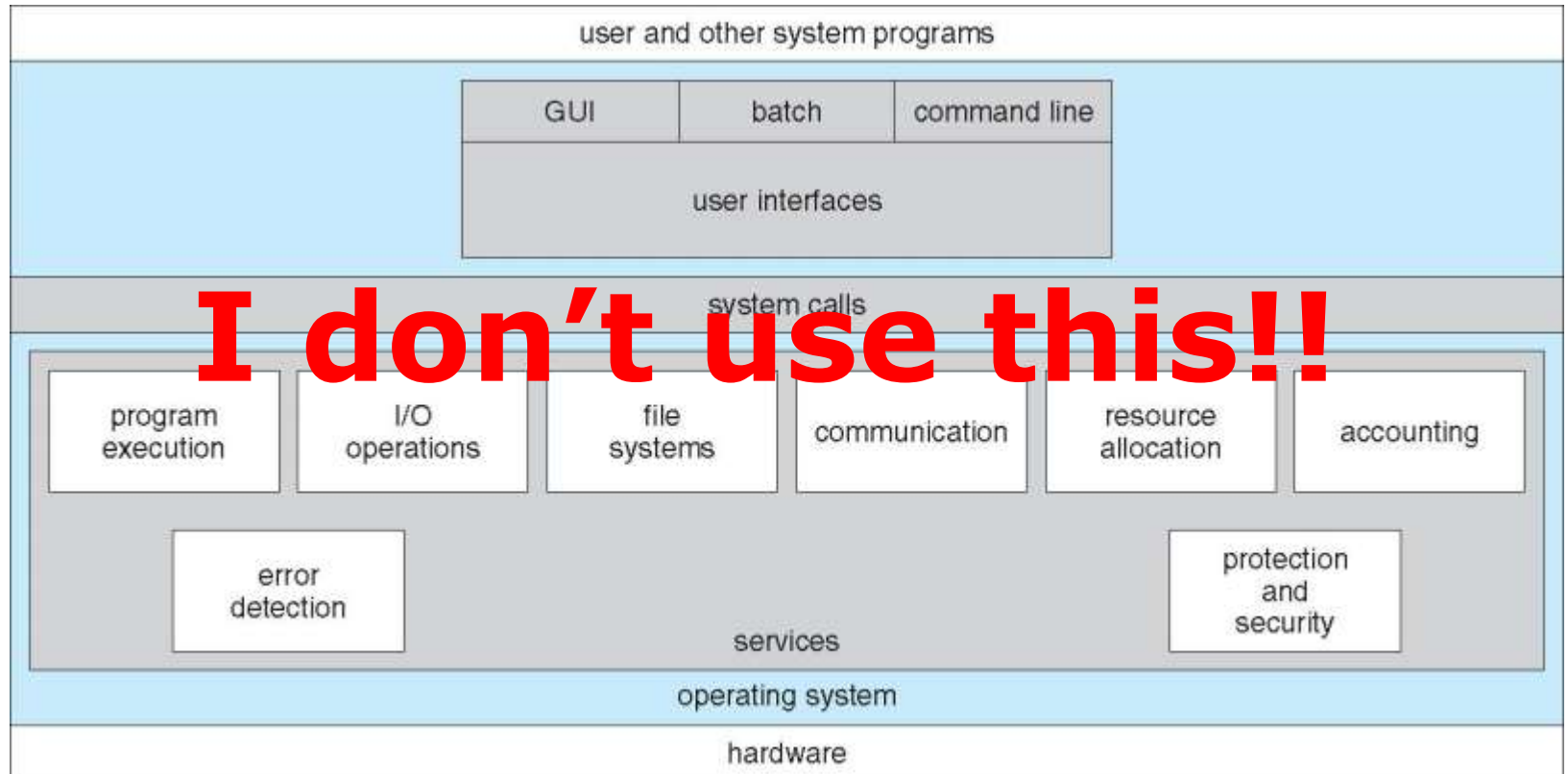
Device
= IO Device
= **External Device**
= Peripheral
= IO Peripheral
= External Peripheral





Four Components of a Computer System

A figure in textbook



[1]



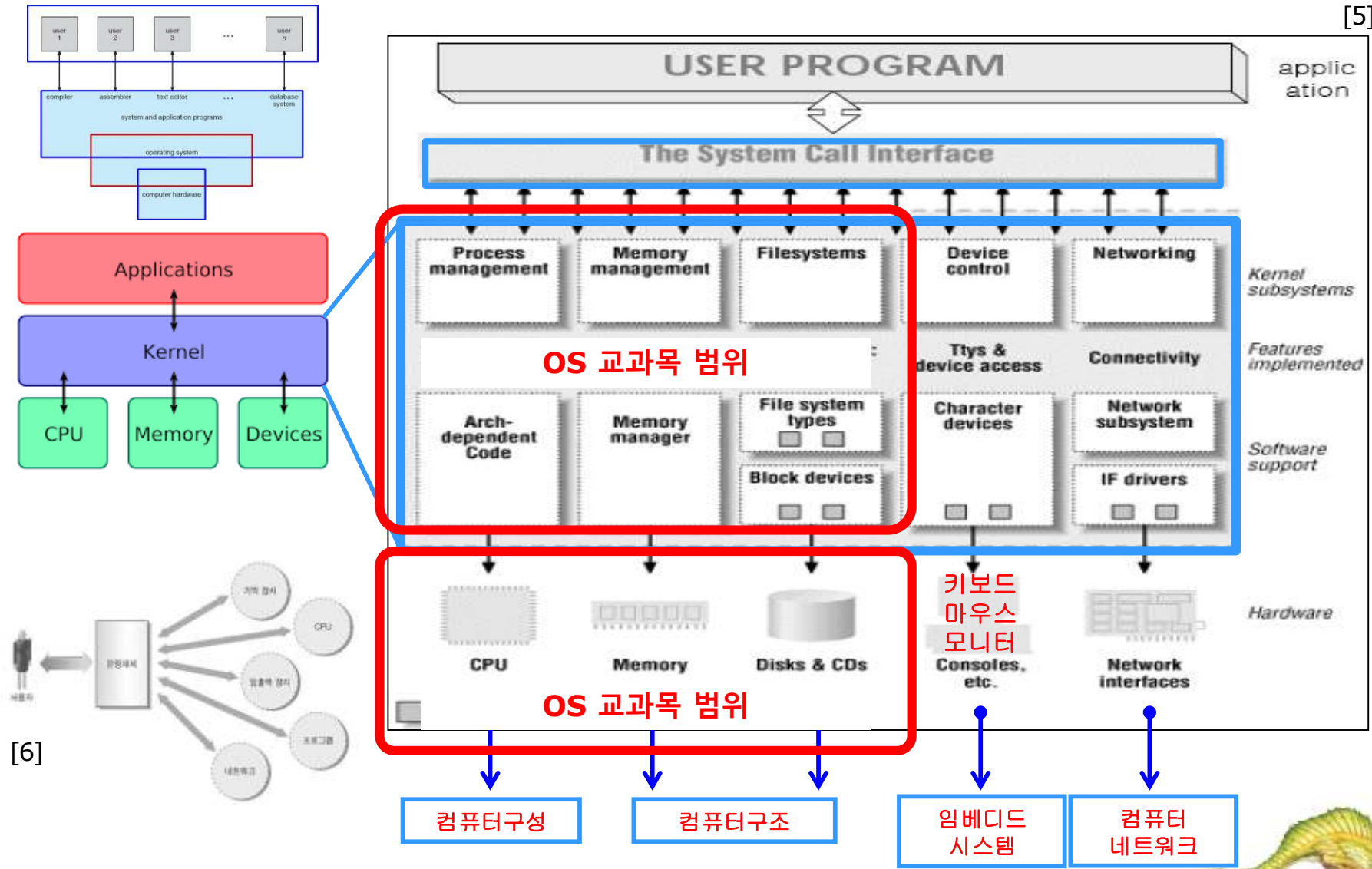


Four Components of a Computer

김규호 (****2008) 님의 모두에게: 오후 5:25
운영체제에서 커널 외의 구성요소가 궁금합니다.
답) 운영체제, 이리온, 윈도우 서비스 등등... 더 구체적인 항목들은
내가 찾아보고 후일기에 올리게드리겠습니다.

출처: <http://www.dreamw.com/20> [소문의 진실]
커널은 컴퓨터가 구동되면서부터 종료하는 시점까지 계속적으로 컴퓨터의 메모리에 남아서 프로세스와 메모리 할당
작업을 수행하게 됩니다.
그 이외의 응용 유틸리티를 이라고 말한 부분은 윈도우즈의 경우로 따진다면 커널을 제외하고 기본적인 작업을 할 수
있도록 OS 패키지에 같이 포함되어 있는 응용 프로그램이라고 할 수 있습니다. 윈도우즈의 경우 커널과 응용 프로그램을 만들기
위한 데오와 그 외의 응용 프로그램을 만들기 위한 7월의 경우, 운영체제 등등이라고 볼 수 있지만,
현재로서는 왜냐하면 윈도우즈나 다른 OS들은 OS들이 커널과 커널 이외 응용 유틸리티가 패키지로 되어 있어서 한 응용은
나오나 OS에 포함된다 라고 생각하는게 일반적이지만 엄밀하게 따졌을 때 OS라고 한다면 응용 유틸리티를
제외한 커널 부분만 해당됩니다.

[5]



[6]






Answer to the Last Question



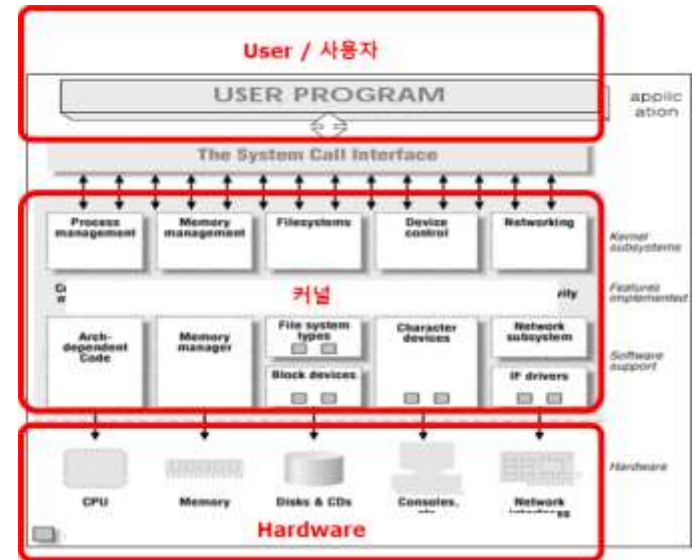
Introductory Questions

- What can you do with computers without Operating Systems?
- What does Operating Systems do?
- What are the purposes of the OS in doing those tasks?
- How the OS performs the tasks to accomplish its purposes (algorithms)?
- Can you implement those algorithms? Then, analyze? Then, design?
- How much the hardware are engaged? 

Operating System Concepts – 9th Edition

1.2

Silberschatz, Galvin and Gagne ©2012



OS is the Hardware Manager & It works for users !!

- It works for users ...
 - who want to build and/or run their programs(software)
 - ▶ On a computer system
 - ▶ which are composed of ❶ CPU(s), ❷ Memory(registers, caches, main memories), ❸ HDD(or SSD), ❹ devices, ❺ network interfaces
- So, you must well understand how computer hardware are working individually as well as collaboratively

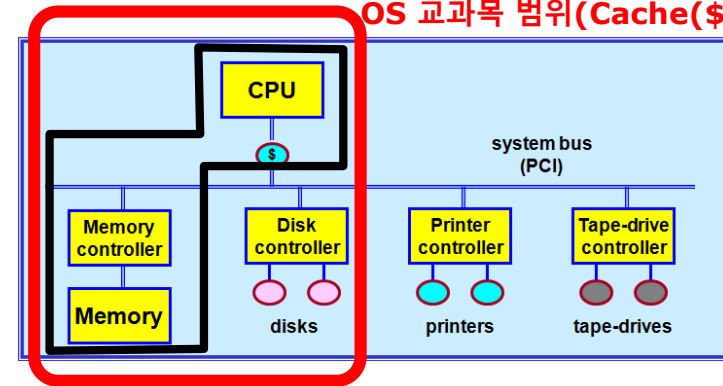
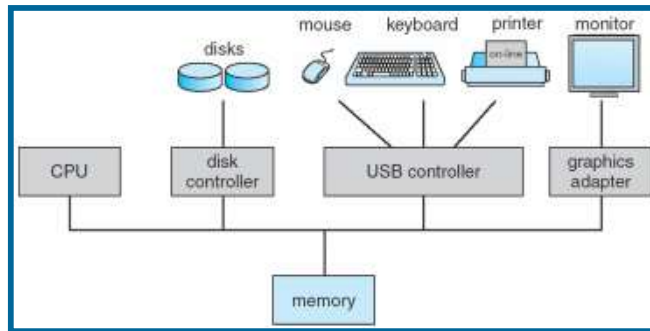




Fundamentals for Computers

- 1. Computer systems consist of **CPU, Memories, HDD & IO Devices + Bus**

[1]



OS 교과목 범위(Cache(\$))제외)

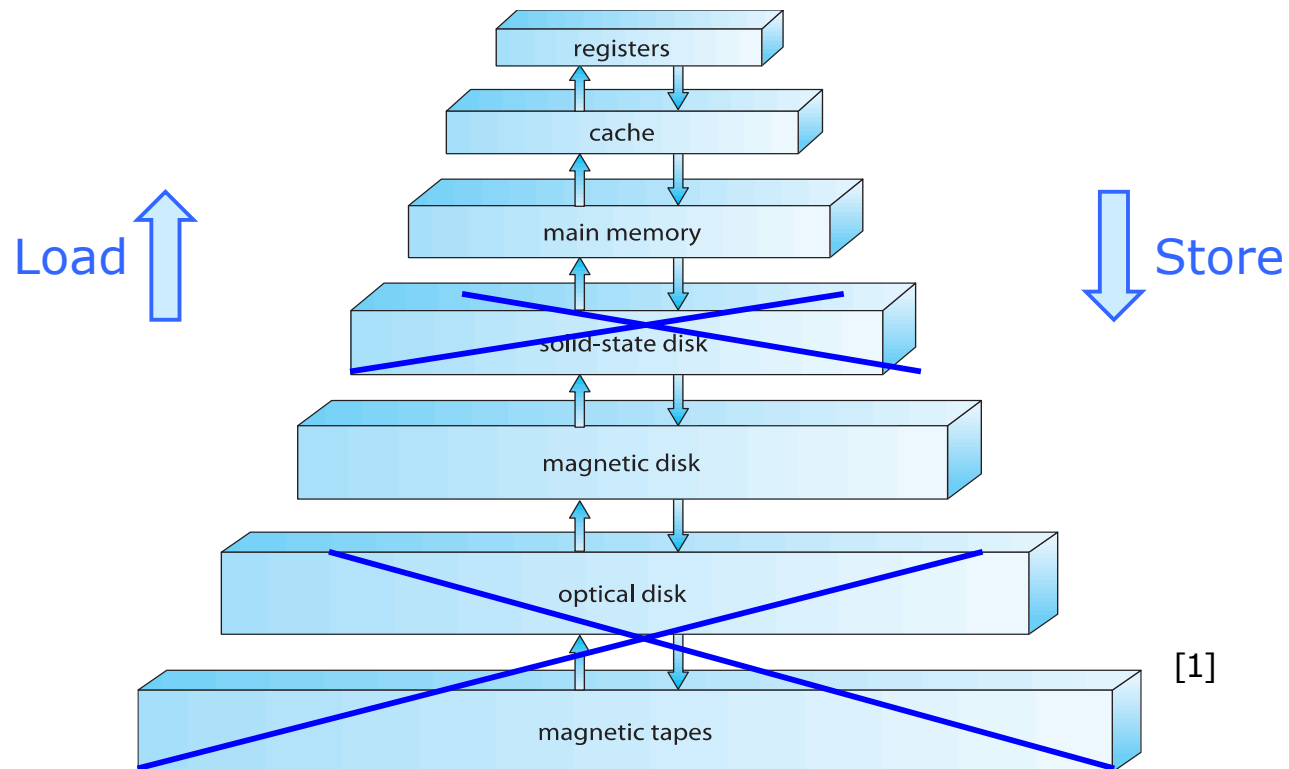
- 2. **CPU & MM** are the only “**internal**” devices
- 3. All others including HDD are “**external**” devices
- 4. **Storages** are classified into two categories
 - “**Internal storages**” such as registers, caches, main memory are **volatile**
 - ▶ Among internal storages, we are **only** interested in **Main memory**, in OS class.
 - “**External storages**” such as HDD, SSD are non-volatile \Rightarrow **permanent**
 - ▶ Among external storages, we are **only** interested in **HDD**, in OS class.
 - ▶ From now on, we only use HDD to represent all permanent storages unless commented otherwise





Fundamentals for Computers

- 5. All files are assumed to be stored in **HDD**
- 6. All programs must be “loaded” from HDD into **MM** to be executed by **CPU**
 - Load : Data movement upward
 - Store : Data movement downward



Memory Hierarchy(메모리 계층구조)

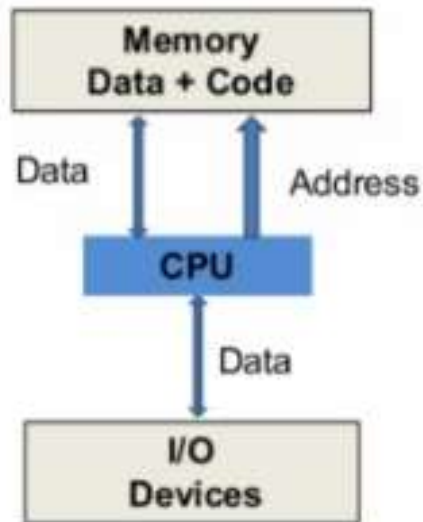
[1]



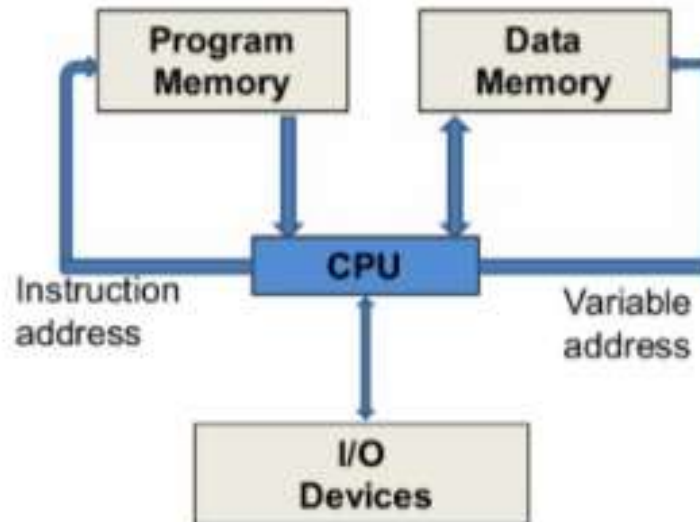


Fundamentals for Computers

- 7. We assume the MM is **Illinois Architecture(= von Neumann Architecture)**
 - cf. Harvard Architecture
 - NOTE : Architecture = Machine Architecture = Computer Architecture



Von Neumann Machine



Harvard Machine

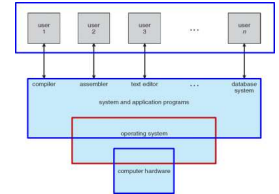
[7]





Fundamentals for Computers

- 8. All “**internal activities**” are **execution of programs** by **CPU** using **Main Memory**
 - That is, if there are **activities** inside computer system, there must be certain **programs** being **executed** by CPU using the Main Memory
- 9. There are only two types of programs
 - **User programs**(& system programs) & **OS programs**
 - We assume **system programs** such as browsers, compilers, editors and so on as **user programs**
- 10. That is, all programs except **user programs** are **OS programs**
- 11. OS performs **Process Management(=Mgt)**, **Resource Mgt** & **Control Programs**
 - Process Managements :
 - ▶ Start to execute, run, stop(exit, pause, kill(=terminate))
 - ▶ Scheduling, Communication, Synchronization, etc.
 - Resources :
 - ▶ Physical resources : CPU, Memories, IO Devices, etc.
 - ▶ Logical resources : Time, Files, etc.
 - Program Control
 - ▶ to prevent errors and improper use of the computer





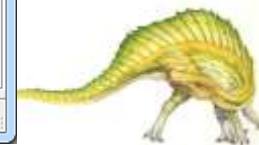
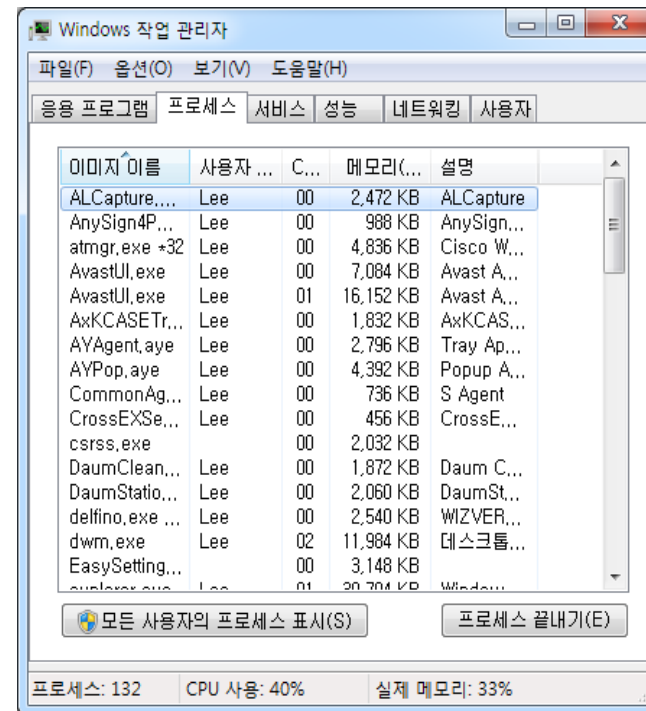
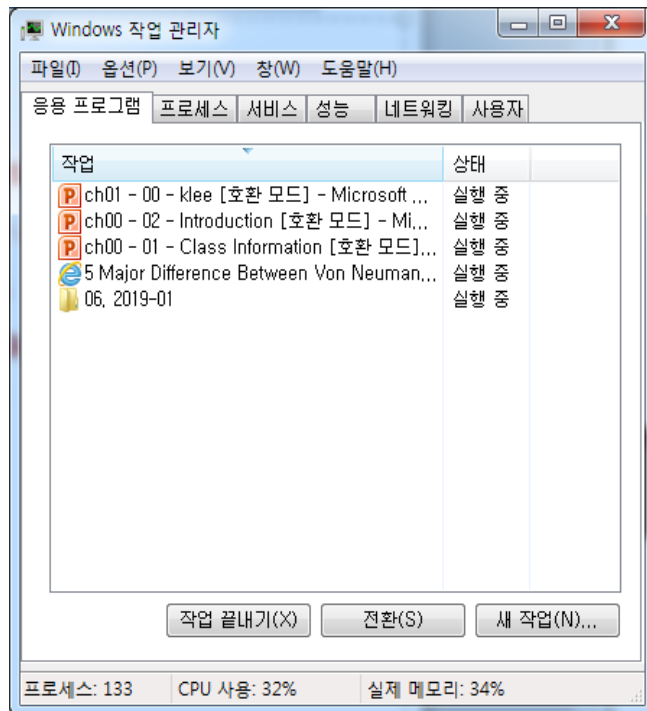
Fundamentals for Computers

김지현(*****2043) 님이 모두에게: 오후 5:30
교수님 user program에 os program 설명하실때
segmentation fault를 계속 들으셨는데,
컴파일 오류는 컴파일러가 처리하는 것으로
알고 있는데 컴파일러는 user program에
숙하지 않나? 왜 os가 처리한다고 하셨는지 궁금합니다!
(답) segmentation fault는 Run Time Error입니다.
참고로, 프로그램은 작성할 때, 프로그램은 단순한 텍스트입니다.
(1) 컴파일 할 때, 에러 확인이 있음. 컴파일 타임 에러!
(2) 이는, 컴파일러가 제공하는 서비스이며,
(3) 실행할 때, 에러 확인이 있음. 런타임 에러!
이는 OS가 제공하는 서비스입니다.

김근호(*****2008) 님이 모두에게: 오후 5:31
cpu에서 메모리에서 불러오는 것보다 하드디스크에서
직접 불러오는 것이 더 느린지 모르겠습니다.
(답) 메모리 속도 = 50~100 nsec (10⁻⁹~10⁻⁸) 10⁹ Hz
HDD 속도 = 10~30 msec (10⁻³~3) 10³ Hz
CPU 속도 = 3GHz 10⁹ Hz

홍정민(*****1994) 님이 모두에게: 오후 5:31
교수님, 배이오스/마이크로소프트 OS 프로그램인가?
(답) 네, 그렇습니다.
다른 하드웨어에 많이 의존하므로, OS 설계기지에서
제공하지 않고, 별도로 컴퓨터 제작/판매사에서
제공합니다.

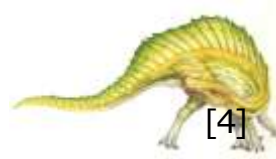
- 12. We assume **single-CPU / multi-user system** in our class
 - Each user may have multiple processes
 - **Therefore, many processes exist, at a time, in a system!**
 - 13. Computer system shared by processes **must keep all users & programs happy**
- Goal** ▶ Happiness is from user convenience, fastness
of OS ▶ For this, OS tries for fair & efficient HW resource management





Questions

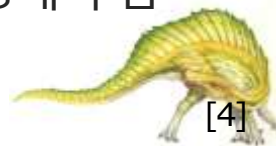
- Q1. 도서관에서 책을 대출할 때, 누구에게 신청하는가?
- Q2.
- Q3. 도서관에서 책을 다 읽은 후, 그 책을 어디에 놓아야 하는가?
- Q4.
- 결론 :





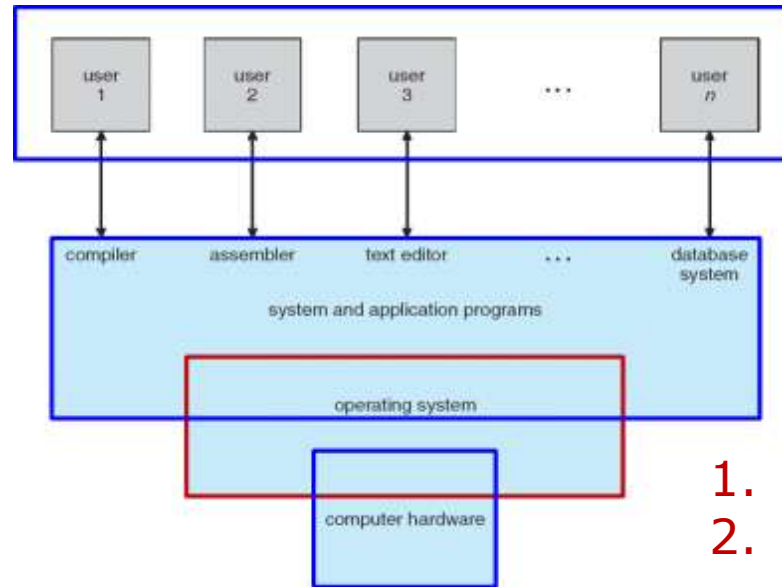
Questions

- Q1. 도서관에서 책을 대출할 때, 누구에게 신청하는가?
⇒ 사서에게 신청함
- Q2. 도서관에서 책을 대출할 때, 왜 사서에게 신청하는가?
⇒ 우리가 원하는 책의 위치를 모르기 때문에
- Q3. 도서관에서 책을 한 권 다 읽은 후, 그 책을 어디에 놓아야 하는가?
⇒ 그대로 책상 위에 두어야 함
- Q4. 도서관 책을 다 읽은 후, 본인이 그 책을 책꽂이에 놓으면 안되는 이유는 무엇인가?
⇒ 모든 책들은 듀이의 분류법에 따라 정렬되어 있으므로,
우리가 잘 못 꽂으면, 다음 사람이 그 책을 찾을 수 없음
- 결론 : 도서관의 책은 많은 사람들이 공유하는 물건이므로
 1. 사용자가 직접 책들을 빼거나(개가식은 가능), 다시 꽂을 권한이 없음!!
 2. 오직 사서(책 “운영자”)만이 책들을 빼거나(개가식 제외), 다시 꽂을 권한이 있음!!
 3. 사용자가 책을 사용하기 위해서는 반드시 사서(책 “운영자”)에게 요청해야 함





Class Rules for OS : Library vs. Computer & OS



이유 / 목적

1. User Convenience
2. Resource Protection

- 14. 도서관의 책은 많은 사용자들이 공유하므로
- 14. 컴퓨터의 하드웨어는 많은 유저들이 공유하므로

1. 사용자가 “직접” 책들을 빼거나, 다시 꽂을 권한이 없음!!

1. 유저가 “직접” 하드웨어를 사용(제어)할 권한이 없음!!

2. 오직 사서(책 “운영자”)만이 책들을 빼거나, 다시 꽂을 권한이 있음!!

2. 오직 운영체제만이 하드웨어를 사용(제어)할 권한이 있음!!

3. 사용자가 책을 사용하려면, 반드시 사서(책 “운영자”)에게 요청해야 함

3. 유저가 하드웨어를 사용하려면, 반드시 운영체제에게 요청해야 함





Fundamentals will be asked in a Surprise Quiz.

You'd better memorize these, all the time!



출처 : <https://blog.daum.net/duaworld/15720106>

14가지 Rules,
믿고 가세요!!!



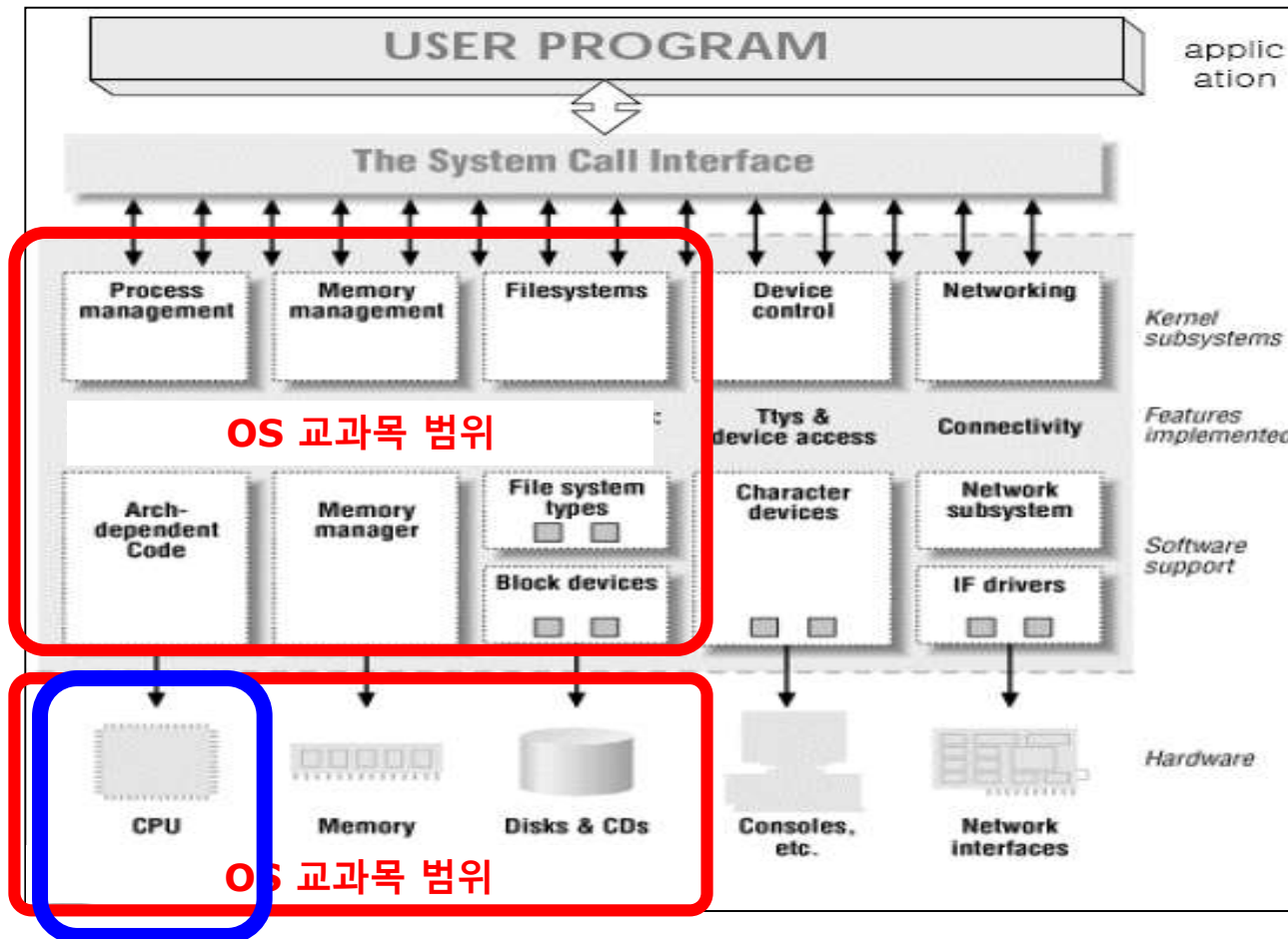
출처 : <https://besuccess.com/opinion/first-step/>





Four Components of a Computer System

[5]

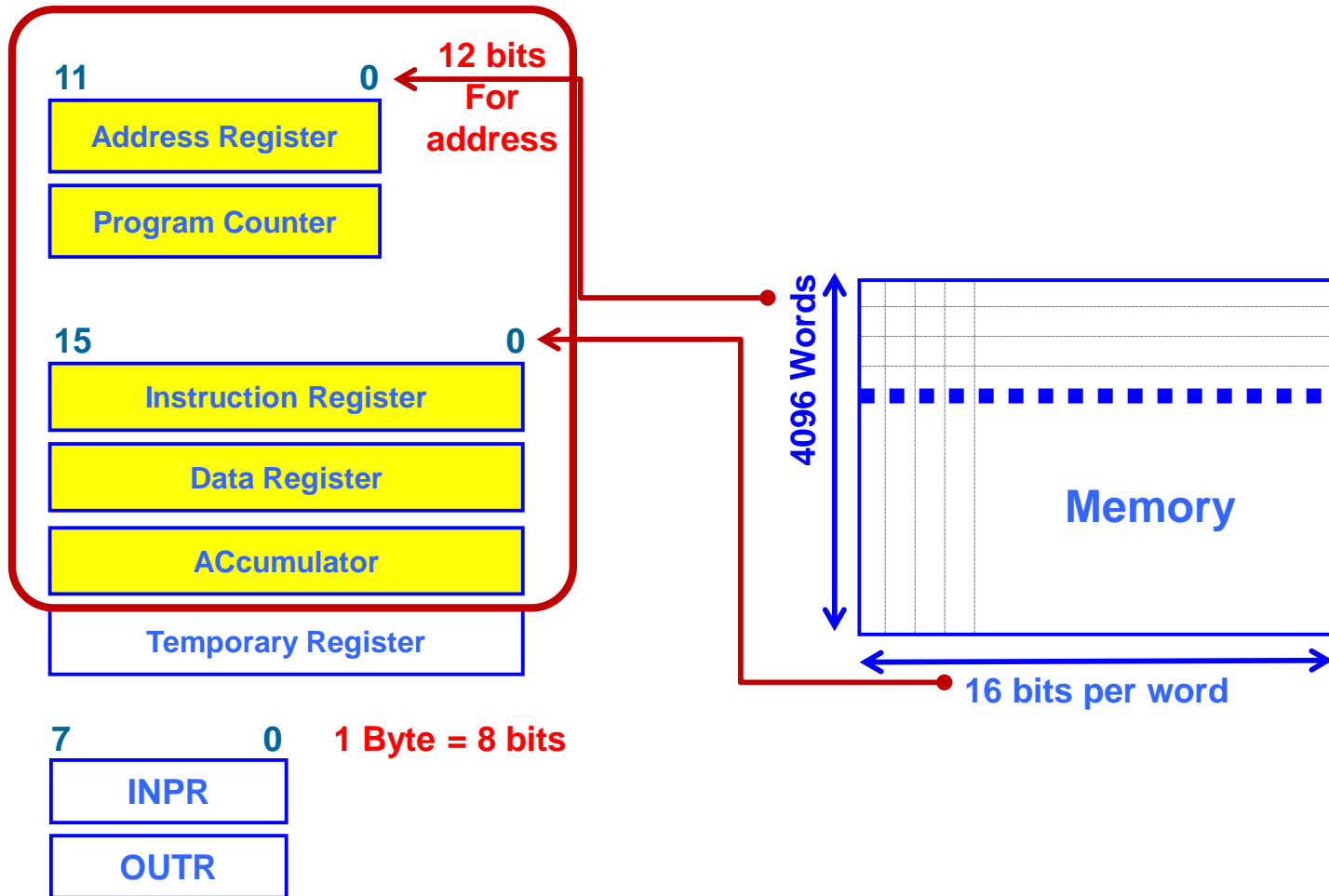


지금부터의
수업





Basic CPU Registers and Memory



[1]





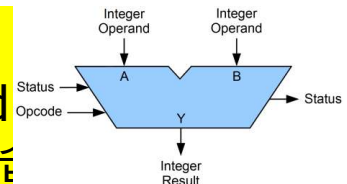
Basic CPU Registers and Memory

(*****2116)임준엽 님이 모두에게: 오후 5:54

교수님 DR에서는 second operand, AC에서는 first operand

했는데 이 first, second가 왜 이렇게 되는지를 정확히 모르겠습

답) 그림에서 보는 바와 같이, 일반적인 ALU는 2개의 데이터(operand)를 입력으로 갖는다. 이 입력을 각각 A, B라고 하면, (1) $A + B$ 와 같이 덧셈이나 곱셈에서는 두 데이터의 순서가 바뀌어도 문제가 되지 않지만, (2) $A - B$ 와 같이 뺄셈이나 나눗셈에서는 두 데이터의 순서가 바뀌면 안된다. 따라서, first operand와 second operand는 명확하게 구분되어야 한다.



(*****2088)김규리 님이 모두에게: 오후 5:55

교수님 저번 시간 피피티 관련해서 질문있습니다. os program은 컴퓨터 시스템 그림에서 어디에 속하는건지 궁금합니다. os program은 사용자도 사용이 가능한데 os시스템에 속하는건가요??

답) 운영체제, Operating System은 하나의 커다란 프로그램이다. (모든 OS의 90% 이상은 C, C++로 짜져 있고, 나머지는 assembler 등으로 구성되어 있다. 따라서, 따라서 OS 프로그램도 운영체제, OS 와 동일한 용어이다.





Basic CPU Registers and Memory

(*****2071)전문수 님이 모두에게: 오후 5:56

교수님 레지스터의 목적에 대한 슬라이드에서 ar이 메모리 주소와 다음 명령의 주소를 저장한다 하셨는데 다음 슬라이드에서는 다음에 수행할 명령의 주소를 저장하는 것은 pc만 한다고 하셔서 혹시 추가로 설명 부탁드립니다 될까요?

답) 중요한 질문입니다. 전문수 군의 말이 맞습니다!! 내가 수업 시간에 강의 자료의 내용을 수정하여 다시 설명을 하도록 하겠습니다.

“다음에 수행할 명령어의 주소는 PC에만 저장된다. 단, 이 PC에 저장된 값이 (메모리의 위치를 지정하기 위하여) AR로 이동된다. 이 때, AR은 자신에게 저장되어 있는 값이 데이터의 주소인지 또는 다음에 수행할 명령어의 주소인지 구분하지 않는다!!.”





Special Purpose Registers in 16 bit CPU / 4KW MM

Name	Description	Usage	Size
AR	Address Register	Holds “memory address” of data or next instruction (단, 다음에 수행할 명령어의 주소는 PC로부터 전달된 값임)	12
PC	Program Counter	Holds “memory address” of “next” instruction	12
IR	Instruction Register	Holds instruction to execute Whole Path of Instruction : HDD ⇐ Memory ⇐ IR ⇐ (instruction) ⇐ Memory ⇐ HDD	16
DR	Data Register	Holds data (= 2 nd operands / results) 1 st Path of Data : DR ⇐ (operand) ⇐ Memory 2 nd Path of Data : DR ⇒ (result) ⇒ Memory	16
AC	Accumulator	Performs Numeric/Logical 1 st Operations & Stores Result (AC ← AC + DR) Whole Path of Data : HDD ⇒ “MM ⇒ DR” ⇒ AC ⇒ “DR ⇒ MM” ⇒ HDD	16

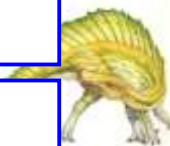
TR	Temporary Register	Holds temporary data	16
INPR	Input Register	Holds input character	8
OUTR	Output Register	Holds output character	8





Rule #15 : for Special Purpose Registers

Name	Rules
AR	Only holds address and no other register does this
PC	Only holds memory address of next instruction and no other register does this (단, PC에 저장된 값이 (메모리의 위치를 지정하기 위하여) AR로 이동된다. 이 때, AR은 자신에게 저장되어 있는 값이 데이터의 주소인지 또는 다음에 수행할 명령어의 주소인지 구분하지 않는다!!
IR	Only holds instruction to execute and no other register does this
DR	Only holds data to/from memory and no other register does this
	The data may be the second operand and no other register does this
AC	Only holds the first operand or result and no other register does this
TR	Only Holds temporary data
INPR	Holds input character and no other register does this
OUTR	Holds output character and no other register does this





Rule #15 : for Special Purpose Registers

(*****2071)전문수 님이 모두에게: 오후 5:56

교수님 레지스터의 목적에 대한 슬라이드에서 ar이 메모리 주소와 다음 명령의 주소를 저장한다 하셨는데 다음 슬라이드에서는 다음에 수행할 명령의 주소를 저장하는 것은 pc만 한다고 하셔서 혹시 추가로 설명 부탁드립니다 될까요?

답) 중요한 질문입니다. 전문수 군의 말이 맞습니다!! 내가 수업 시간에 강의 자료의 내용을 수정하여 다시 설명을 하도록 하겠습니다.

"다음에 수행할 명령어의 주소는 PC에만 저장된다. 단, 이 PC에 저장된 값이 (메모리의 위치를 지정하기 위하여) AR로 이동된다. 이 때, AR은 자신에게 저장되어 있는 값이 데이터의 주소인지 또는 다음에 수행할 명령어의 주소인지 구분하지 않는다!!."

(*****2158)김희수 님이 모두에게: 오후 4:21

교수님. AR에서 address가 메모리로 들어갈때 버스는 사용하지 않는건가요?

답) 네!!

(*****2088)김규리 님이 모두에게: 오후 4:23

교수님 그럼 버스가 아닌 뭐를 통해 전달되는건가요??

(*****2070)송민수 님이 모두에게: 오후 4:23

교수님 그러면 AR에서 버스를 사용하지 않으면 메모리로 들어가는거고 버스를 사용하면 IR로 가는건가요?

(*****2158)김희수 님이 모두에게: 오후 4:25

버스와 메모리의 연결이 있으니 MUX에서 1이 지정되서 address가 AR에서 버스를 통해 메모리로 들어갈 수 있다고 생각했습니다. 만약 address가 버스를 통해 메모리로 들어가는게 아니라면 메모리와 연결된 버스는 어떤 용도가 있나요?

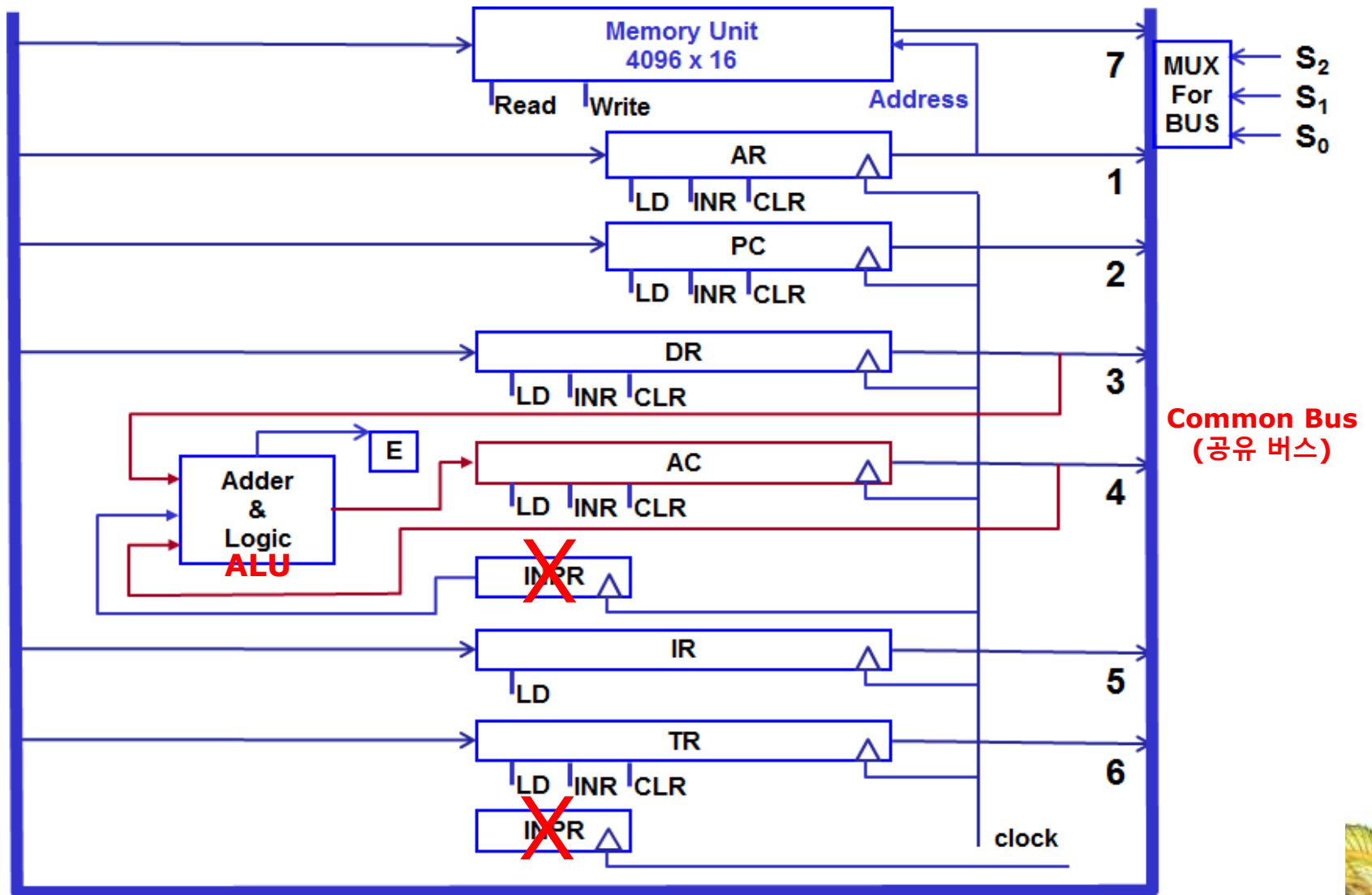
답) 우선, 버스에 대해서 다음과 같이 정의하기로 합니다. "데이터를 전송(송신 및 수신)하기 위한 물리적인 선로(통로, 채널)를 의미함. 직렬 버스(Serial Bus)는 하나의 선로에 (기차와 같은 형식으로) 여러 비트의 데이터를 순차적으로 전송하며, 병렬 버스(Parallel Bus)는 여러 개의 (다발로 이루어진) 선로에 여러 비트의 데이터를 동시에 전송함 "

다음 슬라이드에서 보듯이, 우리가 사용하는 버스는 병렬 버스이며, 통상적으로 지칭하던 큰 디귤자 모양의 병렬 버스의 정확한 명칭은 "공유 버스(Common Bus)"임. 이와는 구분되는 AR과 Memory를 연결하는 선로 역시 12비트를 동시에 전송하는 병렬 버스임. 이는 AR과 Memory를 직접 연결하며, "공유 버스"를 사용하지 않으므로 MUX를 필요로 하지 않음





Data Unit(= Data Path) in CPU

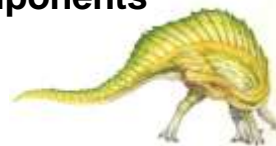
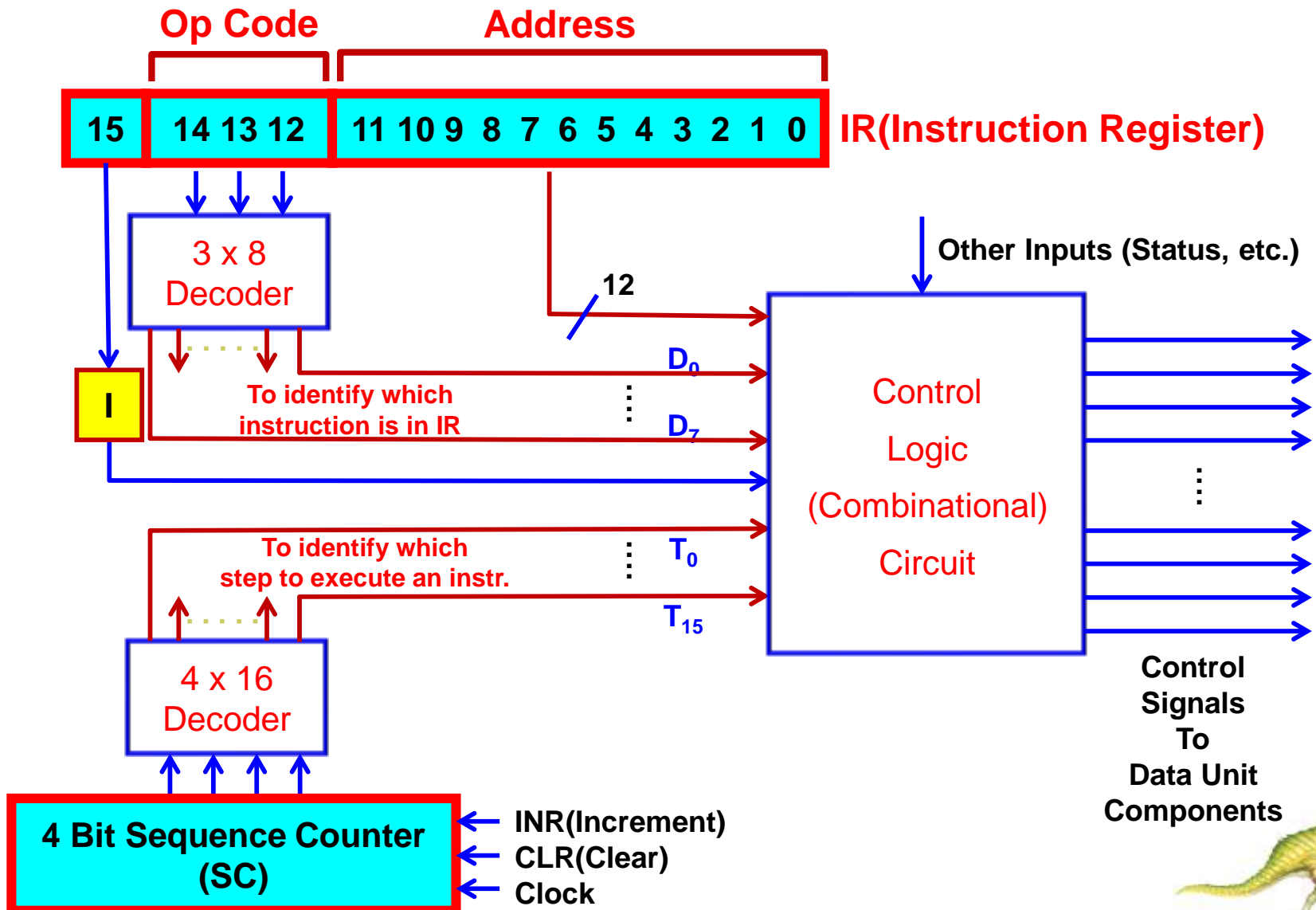


[1]





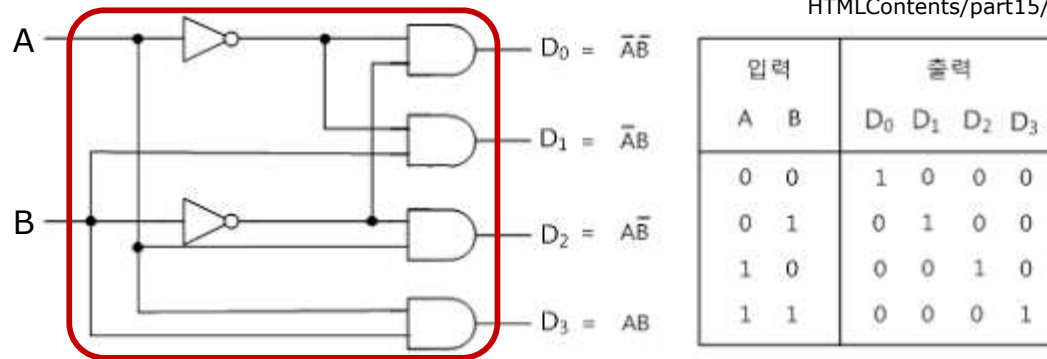
Control Unit in CPU





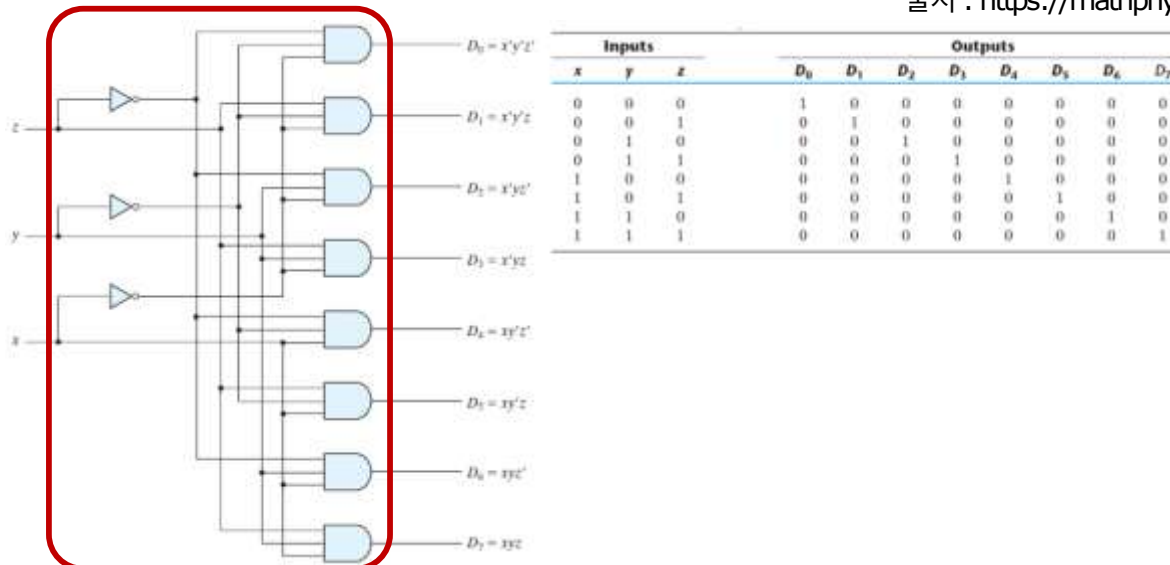
Note : Decoder

출처 : http://ebook.chungpaemt.co.kr/AReS_EO_HTML/HTMLContents/part15/experiment15_1_ko.html



2x4 Decoder : AB(2개)로 구성된 input에 따라 $4(2^2)$ 개의 output 중 하나만 1이 되고 나머지는 0

출처 : <https://mathphysics.tistory.com/583>



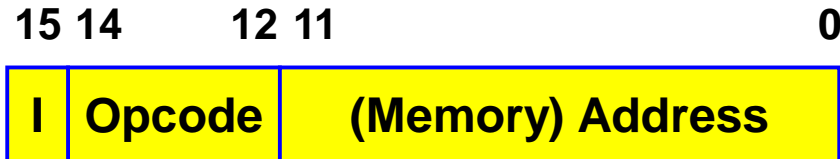
3x8 Decoder : xyz(3개)로 구성된 input에 따라 $8(2^3)$ 개의 output 중 하나만 1이 되고 나머지는 0





Instruction Set Architecture

○ **Memory Reference Instruction** : 메모리에 저장되어 있는 데이터를 사용하는 명령어



OpCode = 000 ~ **110** = $D_0 \sim D_6$ 중 하나만 1, 나머지는 모두 0

○ **Register Instruction** : 레지스터에 저장되어 있는 데이터를 사용하는 명령어



$D_7 = 1$

○ **Input-Output Instruction** : 데이터 입출력 명령어

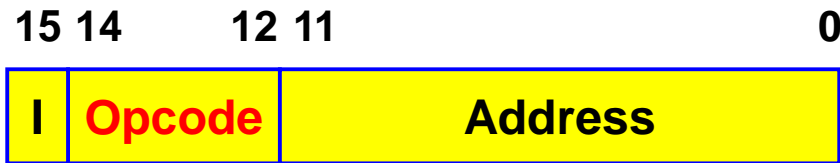


$D_7 = 1$





Memory Reference Instruction Types



Hexadecimal Code

Instruction

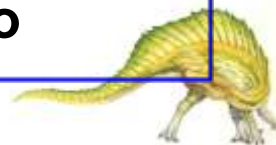
I = 0

I = 1

Description

AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero

xxx : means a certain address





Rule #16 : CPU Instruction Cycle

■ 1. Instruction Fetch

- Bring an instruction from memory to IR
 - ▶ How do we know the address of instruction to execute?

■ 2. Decode

- Identify the opcode of the instruction in IR
 - ▶ What do we make use of to identify the opcode?

■ 3. Decide ~~Effective~~ Address, if MRI instruction

- Read the effective address from memory if 'I' bit = 1
 - ▶ How do we know the address of the operand?
- Read the address from IR

■ 4. Data Fetch, if necessary, if MRI instruction

- Read the data from the target memory location into DR
- Read data from target memory location(=designated by the “address”) into DR

■ 5. Execute

- Run the decoded instruction as control unit operates
 - ▶ Do you know how to implement the control unit?

■ 6. Store Result Data, if necessary

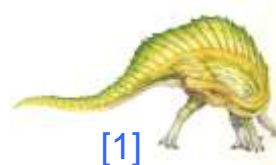
- Write the result data from AC to the target memory location
- Write result data from AC to target memory location(=designated by the “address”)
- How do we know the address of the result?





Rule #16 : CPU Instruction Cycle

- **1. Instruction Fetch**
 - Bring an instruction from memory to IR
 - ▶ How do we know the address of instruction to execute?
- **2. Decode**
 - Identify the opcode of the instruction in IR
 - ▶ What do we make use of to identify the opcode?
- **3. Decide Address, if MRI instruction**
 - Read the address from IR
- **4. Data Fetch if MRI instruction**
 - Read data from target memory location(=designated by the “address”) into DR
- **5. Execute**
 - Run the decoded instruction
- **6. Store Result Data**
 - Write result data from AC to target memory location(=designated by the “address”)





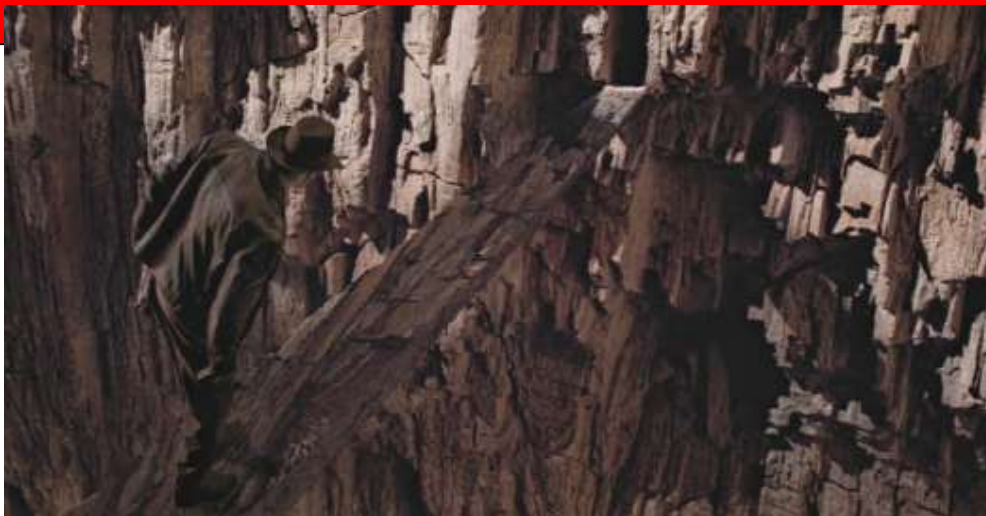
Rule #16 : CPU Instruction Cycle

CPU는 주어진 instruction cycle만 반복적으로 수행한다.
다른 일은 안한다.

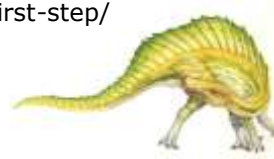
CPU는 전기가 꺼지면 아무 일도 안한다.

CPU는 전기가 켜지면 어떠한 일을 하는가?
(답) CPU는 주어진 instruction cycle만 반복적으로 수행한다.
다른 일은 안한다.

컴퓨터에 전원이 들어왔을 때, CPU가 가장 먼저하는 일은?
(답) Instruction Fetch!!



출처 :
<https://besuccess.com/opinion/first-step/>





1. Instruction Fetch

CPU는 주어진 instruction cycle만 반복적으로 수행한다.
다른 일은 안한다.

CPU는 전기가 꺼지면 아무 일도 안한다.

CPU는 전기가 켜지면 어떠한 일을 하는가?
(답) CPU는 주어진 instruction cycle만 반복적으로 수행한다.
다른 일은 안한다.

컴퓨터에 전원이 들어왔을 때, CPU가 가장 먼저하는 일은?
(답) Instruction Fetch!!

컴퓨터에 전원이 들어왔을 때, CPU가 가장 먼저하는 일은?
(답) Instruction을 fetch하기 위하여 $AR \leftarrow PC$ 를 실행한다!
왜?? PC는 메모리에 연결되어 있지 않음!



심재혁(*****1999) 님이 모두에게: 오후 5:36
교수님, PC도 레지스터의 일종이고 레지스터는 휘발성이라
컴퓨터를 끄는 순간 데이터가 다 날라가는데,
컴퓨터를 키면 주소값이 없어진 PC에서
어떻게 AR에게 주소를 넘겨줄 수 있는 건가요?

답) Power-On 때, 인 순간에 PC에는 어떤 값이 저장되어 있나요?

→ "0000...00"이 저장되어 있습니다.

→ (배터리 방전 상태) "0"이라는 값이 저장된 것이 아니라,
해당 셀에 전하기 없다는 의미!

→ 즉, power-off 동안에, register, cache, MM에는 모두 "0"이
저장되어 있음!

따라서, $AR \leftarrow PC$ 가 실행되고 나면, AR에는 0이 저장됨!

➔ 이는 컴퓨터에서 매우 중요한 의미를 갖는데... 더 중요한 질문은
그러면, 메모리 "0"번지에도 모두 "000...0"이 저장되어 있잖아요.
이런 경우, 컴퓨터/CPU는 무슨 일을 한다는 거죠???

➔ 이 문제에 대해서는, 나중에 Bootloader에서 다루겠습니다!





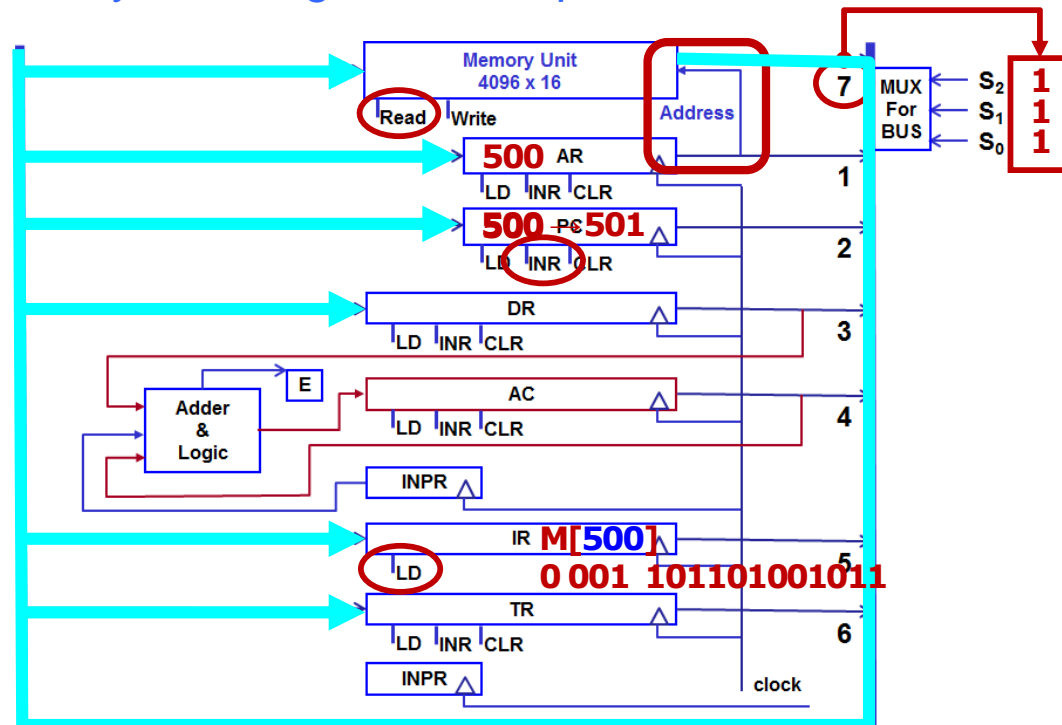


1. Instruction Fetch

Fetch an instruction : Bring an instruction from memory to IR

■ $T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$

- Enable the read input of the memory
- Place the content of memory onto the bus \rightarrow by making $S_2S_1S_0 = 111$
- Transfer the content of the bus to IR \rightarrow by enabling the LD input of IR
- Increment PC \rightarrow by enabling the INR input of PC



[1]





Ex : Register xfers for Fetch phase

T₀: AR ← PC

- Place the contents of PC onto the bus
 - by making the bus selection inputs
- Transfer the contents of the bus to AR

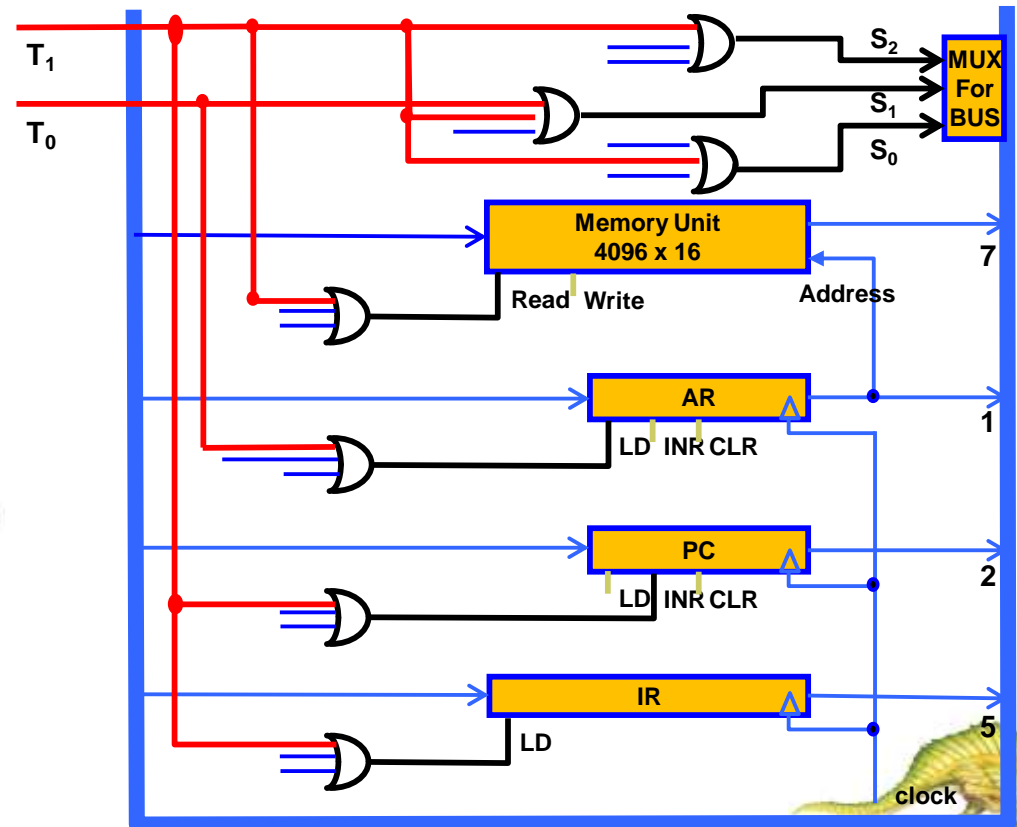
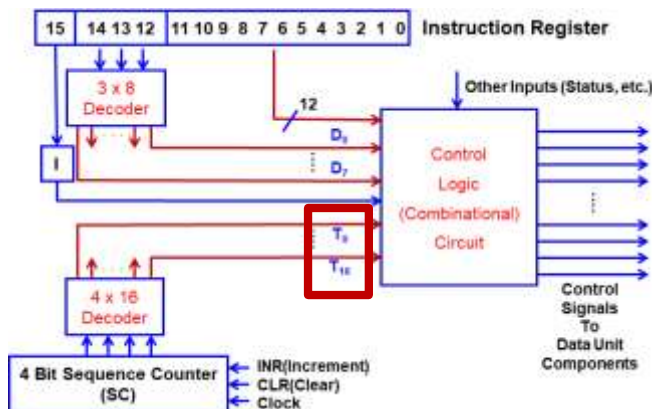
$S_2 S_1 S_0$ to 010

$\Rightarrow S_2 = 0, \underline{S_1 = 1}, S_0 = 0$

T₁: IR ← M[AR], PC ← PC + 1

- Enable the read input of the memory
- Place the content of memory onto the bus
 - by making $S_2 S_1 S_0 = 111$
- Transfer the content of the bus to IR
 - by enabling the LD input of IR
- Increment PC
 - by enabling the INR input of PC

$\Rightarrow S_2 = 1, \underline{S_1 = 1}, S_0 = 1$



[1]





3. Decide Effective Address

■ Decide Effective Address

- If 'I' bit = 1, read the effective address from memory
 - ▶ otherwise, do nothing

■ In T_3 phase, following 4 cases are possible depending on I & D_7 values

■ $D_7'I'T_3$: $AR \leftarrow M[AR]$ ☞ Indirect addressing

- When $D_7 = 0$ & $I = 1$ & $T_3 = 1$
 - ▶ When one of $D_0 \sim D_6 = 1$ ☞ Memory Reference Instruction

Assume :

$M[AR] = IR = 0\ 001\ 101101001011$
⇒ "Add" AC with data in 101101001011
& storer result in AC

■ $D_7'I'T_3$: Do Nothing ☞ Direct addressing

- When $D_7 = 0$ & $I = 0$ & $T_3 = 1$
 - ▶ When one of $D_0 \sim D_6 = 1$ ☞ Memory Reference Instruction

■ Fetch $D_7'I'T_3$: Execute a register-reference instruction

- $D_7 = 1$ & $I = 0$ & $T_3 = 1$ ☞ Register Instruction

■ Fetch D_7IT_3 : Execute an input-output instruction

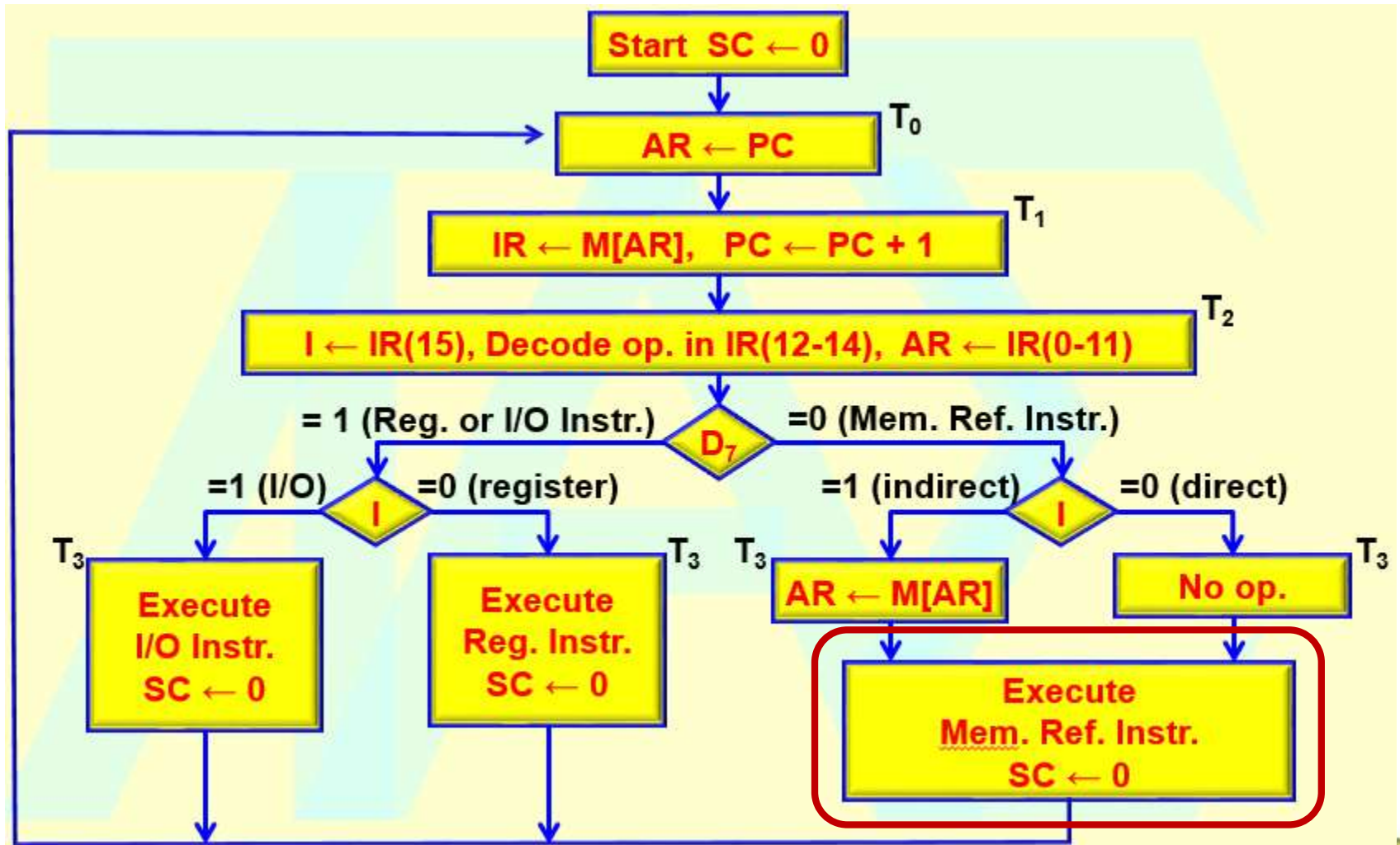
- $D_7 = 1$ & $I = 1$ & $T_3 = 1$ ☞ Register Instruction

[1]





Summary



[1]





4. Data Fetch if MRI

■ Data Fetch for Memory Reference Instruction

- Bring the data from the Memory location pointed by ... to ...
- For other instructions, do nothing
 - ▶ Register instruction & IO instruction

■ Data Fetch & Execute

- D. Fetch : D_0T_4 : $DR \leftarrow M[AR]$ for AND operation
- Execute : D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

- D. Fetch : D_1T_4 : $DR \leftarrow M[AR]$ for ADD operation
- Execute : D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$

- D. Fetch : D_2T_4 : $AC \leftarrow M[AR]$ for LDA operation
- Data Fetch = Execute Load Instruction → Do nothing





5. Execute & 6. Store

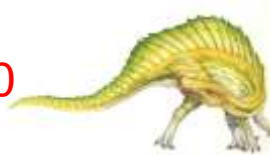
■ Execute Instruction according to the Op-code...

- Perform the coded action while using the data loaded into DR

■ Data Fetch & Execute

- D_0T_4 : $DR \leftarrow M[AR]$ for AND operation
 - ▶ Execute : D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
- D_1T_4 : $DR \leftarrow M[AR]$ for ADD operation
 - ▶ Execute : D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
- D_2T_4 : $AC \leftarrow M[AR]$ for LDA operation
 - ▶ Data Fetch = Load
- Execute = Store : D_3T_4 : $M[AR] \leftarrow AC$, $SC \leftarrow 0$ for STA operation
- Execute : D_4T_4 : $PC \leftarrow AR$, $SC \leftarrow 0$ for BUN operation
- Execute : D_5T_4 : $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$ for BSA operation
 - ▶ D_5T_5 : $PC \leftarrow AR$, $SC \leftarrow 0$
- Execute : D_6T_4 : $DR \leftarrow M[AR]$ for ISZ operation
 - ▶ D_6T_5 : $DR \leftarrow DR + 1$
 - ▶ D_6T_6 : $M[AR] \leftarrow DR$, if $(DR = 0)$ then $PC \leftarrow PC + 1$, $SC \leftarrow 0$

[1]





Summary of Executing d-ADD Op.

■ 1. PC was loaded by the OS

- When user double-click an icon of a program
- 1.1 : The OS finds “the file” from the HDD (Chap. 10, 11)
- 1.2 : The OS finds “empty” space of MM & move it there (Chap. 8, 9)
 - ▶ OS Knows the address of the first instruction of the program
- 1.3 : OS loads the PC = address of the first instruction of the program
 - ⇒ PC keeps the address of the first line of the program

■ 2. CPU moves the value in PC into AR ($AR \leftarrow PC$)

■ 3. CPU moves the contents of M[AR] into IR ($IR \leftarrow M[AR]$)

- During the meanwhile, CPU increments the PC ($PC++$)

■ 4. CPU decodes the instruction stored in IR (using decoder)

- And, CPU moves the address of data to AR ($AR \leftarrow IR(0-11)$)

■ 5. CPU moves data from MM into DR ($DR \leftarrow M[AR]$)

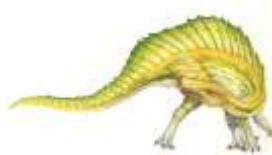
■ 6. CPU executes ADD op using adder & AC ($AC \leftarrow AC + DR$)





자료 출처 / 참고 문헌

- [1] Operating System : Concepts 9th Ed. – By A. Silberschatz, *et al.*
- [2] 위키백과
[https://ko.wikipedia.org/wiki/%EC%BB%A4%EB%84%90_\(%EC%BB%B4%ED%93%A8%ED%8C%85\)](https://ko.wikipedia.org/wiki/%EC%BB%A4%EB%84%90_(%EC%BB%B4%ED%93%A8%ED%8C%85))
- [3] <https://p3ace.tistory.com/34>
- [4] https://www.oss.kr/info_techtip/show/f1c6db27-7caf-44b9-97bd-b4a021e0e5f2
- [5] <https://www.slideshare.net/UbuntuKorea/2010y01m30d>
- [6] <https://ejrtmtm2.wordpress.com/page/2/>
- [7] <https://vivadifferences.com/5-major-difference-between-von-neumann-and-harvard-architecture/>
- [8] Computer System Architecture, 3rd Ed. – by M. M. Mano





QnA

(*****2067)김민성 님이 모두에게: 오후 5:29

교수님 컴퓨터 환경설정은 os program인가요?

답) 네!! 컴퓨터를 관리하는 측면이죠. 즉, OS 프로그램입니다.

(*****2073)이태규 님이 모두에게: 오후 5:30

교수님 특정 프로그램이 로드를 거쳐 cpu에 의해 실행된다면 로드 명령은 어떻게 실행되나요?

답) 로드 : 파일이 HDD에서 MM으로 이동하는 "activity" → 어떤 프로그램이 실행되었다는 의미 → 이 프로그램을 당신이 짰나요? → no! 그렇다면 이 일도 OS가 하는 일이고, 이런 일을 하는 subroutine을 "loader"라고 부릅니다. Memory Management 영역에 포함됩니다.

(*****2006)김건우 님이 모두에게: 오후 5:33

교수님 user program과 OS program의 차이를 구분지을 수 있는 명확한 기준이 있을까요?

답) Sorry!! 수업을 통해서 조금 더 알아보면서 경험적으로 지식을 쌓아 나가도록 합시다.

흐음... 사용자 프로그램은, "사용자가 의도적으로 요구하는 특정한 기능을 제공하는 프로그램. 단, 하드웨어를 직접적으로 사용하거나 관리하는 프로그램은 제외" 정도로 이해를 하고 있습니다.

이런 측면에서 본다면, 브라우저, 컴파일러, DBMS 등도 (사용자) 앱(어플리케이션)에 속한다는 사실을 기억해주시기 바랍니다.





QnA

(*****2073)이태규 님이 모두에게: 오후 5:34

아 교수님 그렇다면 로더의 액티비티는 로드 과정을 거치지 않고 실행되는 것인가요?

답) 유저는 운영체제에게 "특정 파일을 사용하겠으니, 그 파일을 하드에서 찾아서 메모리에 올려다 놓아 달라"는 요청을 합니다. 그러면 OS는 이 일을 loader라는 subroutine을 실행함으로써 유저를 위한 서비스를 수행합니다.

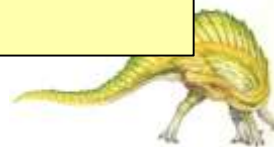
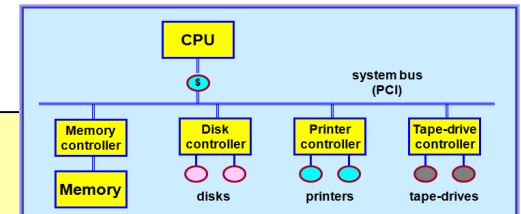
특히, loader는 컴퓨터를 부팅할 때에 가장 먼저 HDD에서 MM으로 올라가는데, 이러한 역할은 뒤에 나올 예정인 boot loader에 의해서 수행됨을 확인할 수 있을 것입니다.

이서연(*****0120) 님이 모두에게: 오후 5:39

교수님 1번 rule에서 말하는 bus는 무엇인가요?

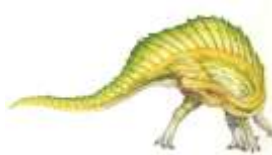
답) 버스는 다양한 하드웨어 장치들을 연결시켜주는 일종의 전기줄입니다. 이 전기줄을 통해서 데이터/프로그램이 이동합니다.

특히, 이 그림에서 버스는 "병렬(parallel) 버스"이며, 총 32개 또는 64개의 전기줄묶음으로, 한 줄에 한 비트씩, 그러니까 한 번에 32비트 또는 64비트씩 전송되는 경로입니다.





Any Question?





참고 자료 : Memory in 컴퓨터 구성

In Random-Access Memory the memory cell can be accessed for infor. transfer from any random location

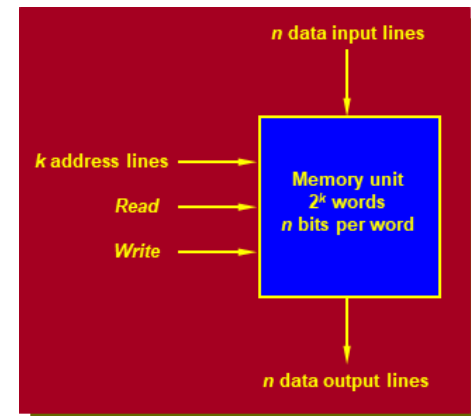
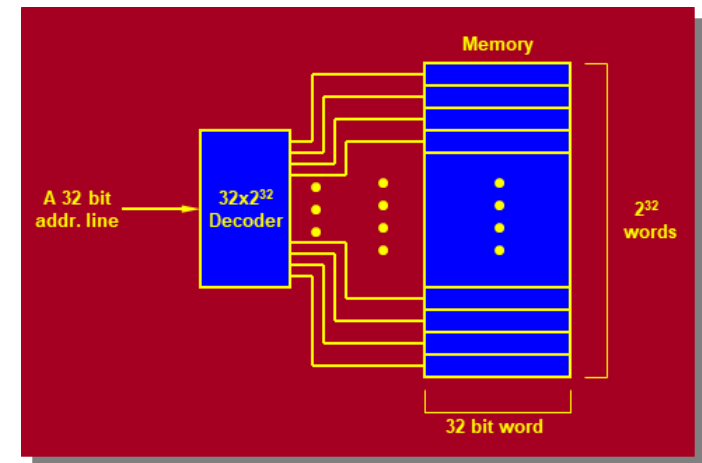
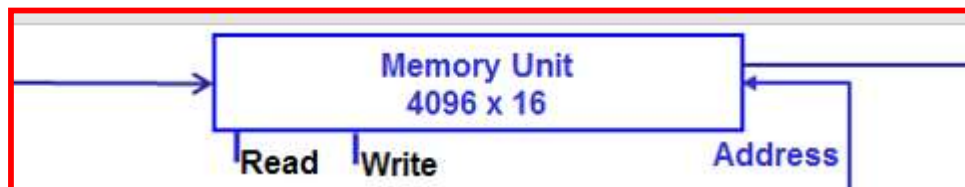
➤ A process locating any word in memory takes the same amount of time

❖ Store (Write) Operation

- ① Apply desired binary addr. into **addr. line**
- ② Apply **data bits** into data input line
- ③ Activate the **write** line

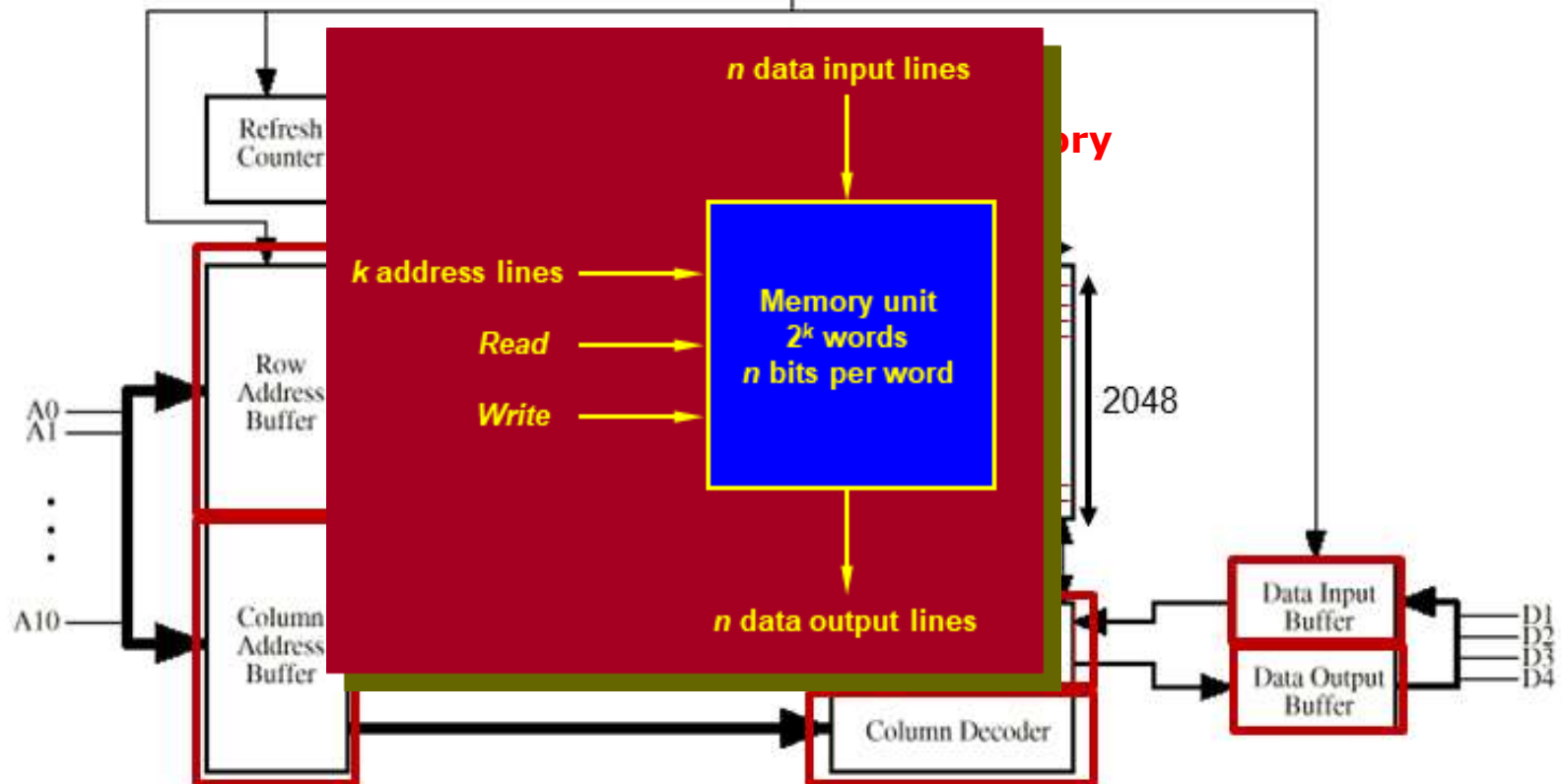
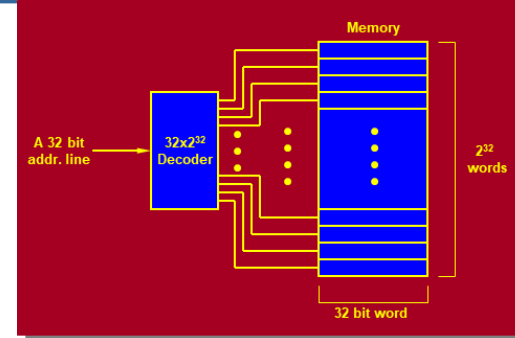
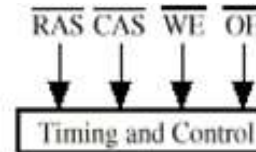
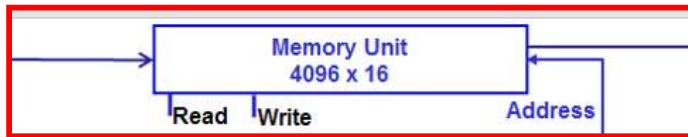
❖ Load (Read) Operation

- ① Apply desired binary addr. into **addr. line**
- ② Activate the **read** line





참고 자료 : Memory in 컴퓨터 구조

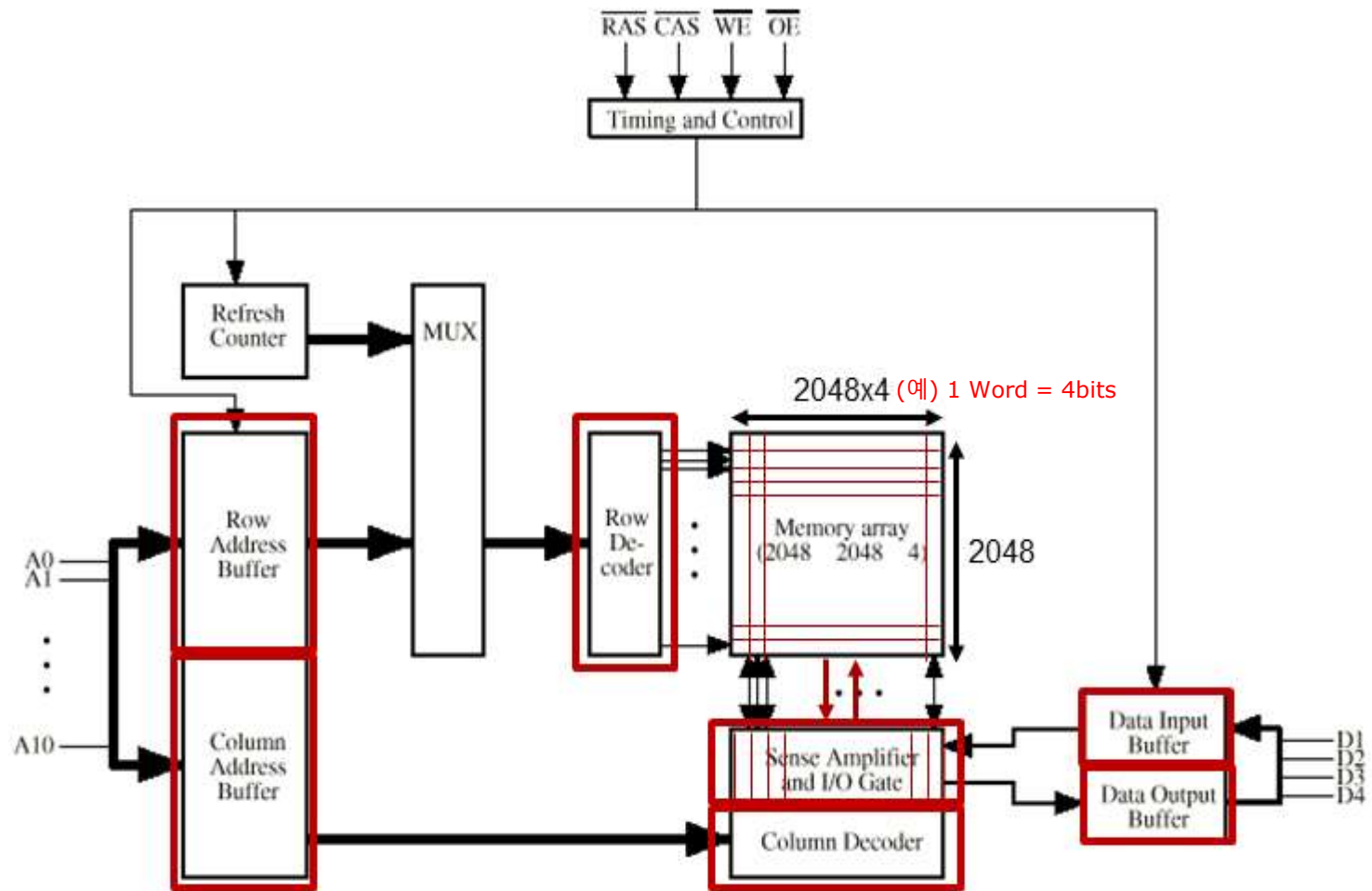


[2]





참고 자료 : Memory in 컴퓨터 구조



[2]





참고 자료 : ALU 구조

Half Adder : 반가산기(Carry 입력을 받지 않음)

Truth Table

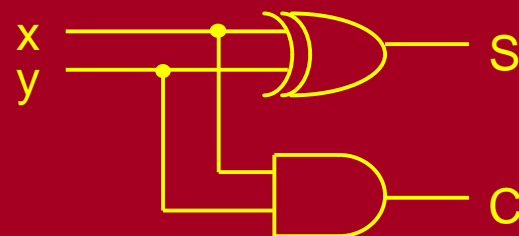
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Boolean Function

$$S = x'y + xy' = x \oplus y$$

$$C = xy$$

Logic Diagram





참고 자료 : ALU 구조

Full Adder : 전가산기(Carry 입력을 받음)

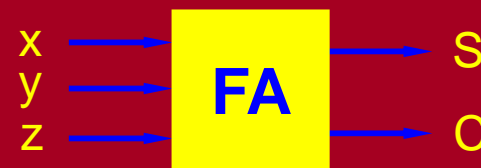
Truth Table

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

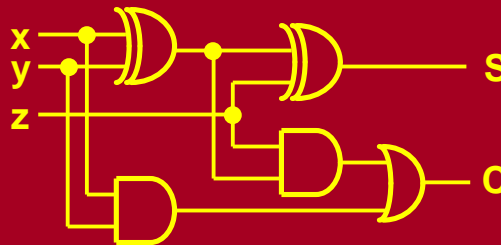
Boolean Function

$$S = x \oplus y \oplus z$$
$$C = xy + (x'y + xy')z$$
$$= xy + (x \oplus y)z$$

Block Diagram



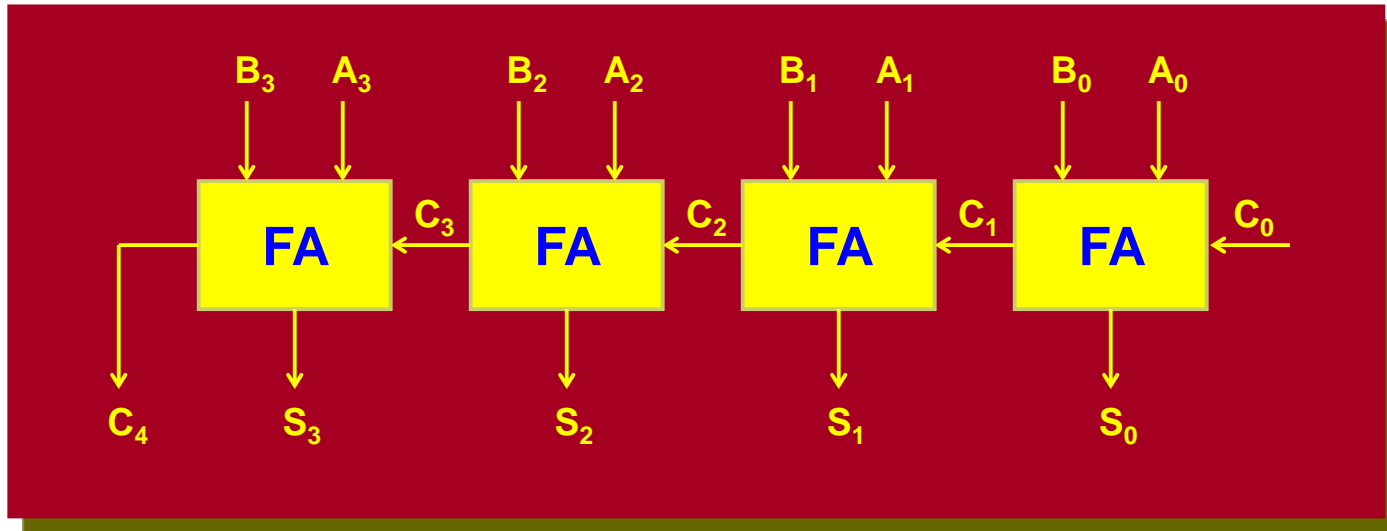
Logic Diagram



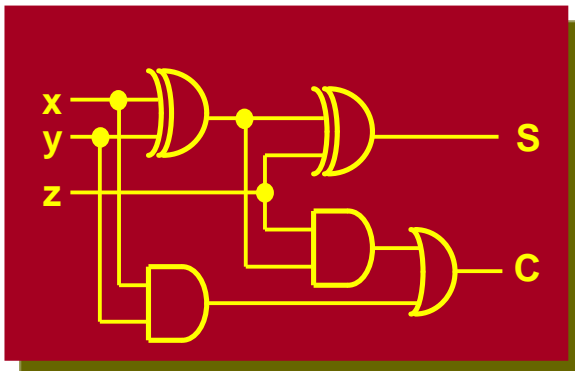


참고 자료 : ALU 구조

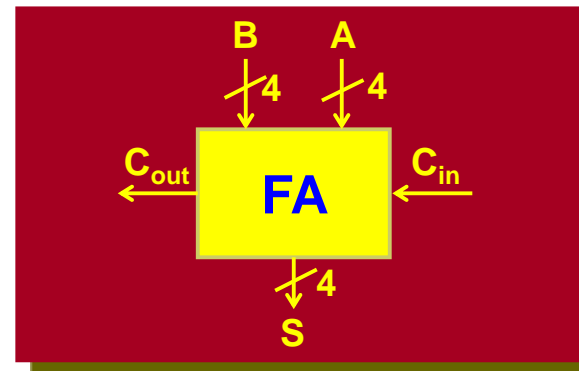
4 Bit Full Adder



Full Adder



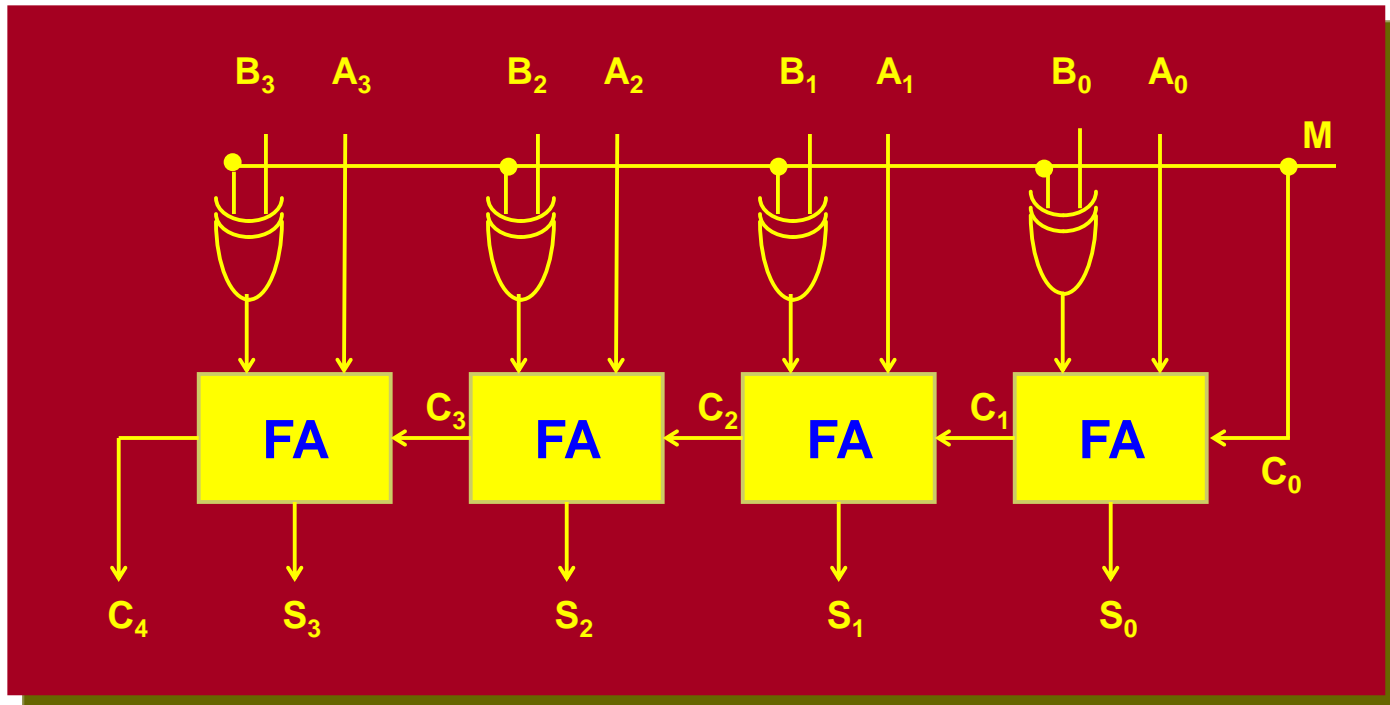
Block Diagram





참고 자료 : ALU 구조

4bit Adder-Subtractor



If $M = 0 \Rightarrow A + B$

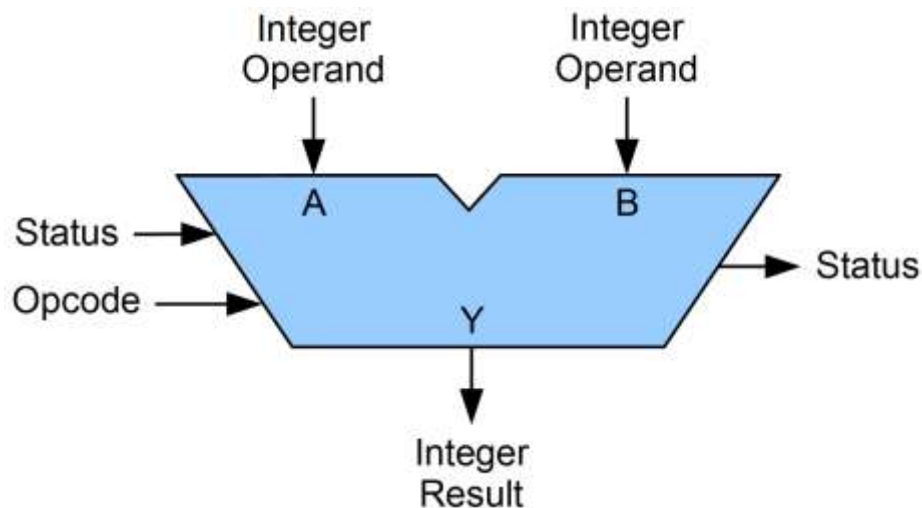
If $M = 1 \Rightarrow A + (B' + 1)$
 $\Rightarrow A - B$





참고 자료 : ALU 구조

Arithmetic & Logic Unit



Arithmetic Operations :

- Adder, Subtractor, Multiplier, Divider, Shifter 등

Logical Operations :

- AND, OR, NAND, NOR, XOR, NOT, MASK 등

