



프로그래밍 원리와 실습 주제 1

# 컴퓨터 시스템과 c 언어 소개



부산대학교 정보·의생명공학대학  
정보컴퓨터공학부



부산대학교  
PUSAN NATIONAL UNIVERSITY

# 강의 목표와 구성

## ❖ 컴퓨터 시스템과 프로그램의 이해

- Universal Computing Device and Turing Machine
- 컴퓨터 시스템 : H/W + S/W
- Sequence of Instructions (명령어 순차 조합)
- 컴퓨터를 활용한 문제 해결 :  
Transformations between layers of abstraction
- Algorithm의 속성

## ❖ C 언어 소개와 컴파일의 이해

- 프로그래밍 언어 분류
- C 언어 소개
- Compiled Language vs. Interpreted Language
- C 언어 컴파일과 Python 언어 인터프리터 비교하기
- Von Neumann Architecture

# 컴퓨터 시스템과 프로그램의 이해

# Introduction to the World of Computing

## ❖ Computer: **electronic genius**?

- NO! **Electronic idiot!**
- Does exactly what we tell it to, nothing more.

## ❖ Hardware vs. Software

- It's not either/or – **both are components of a computer system.**
- Even if you specialize in one, you should understand capabilities and limitations of both.

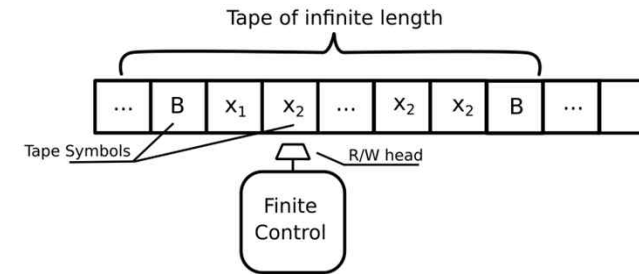
## ❖ Computer is a **Universal Computing Device**

- It is not limited to perform only a specific computation, it can perform any computation
- The idea of a universal computational device is due to Alan Turing.



# Turing Machine

컴퓨터의 공간

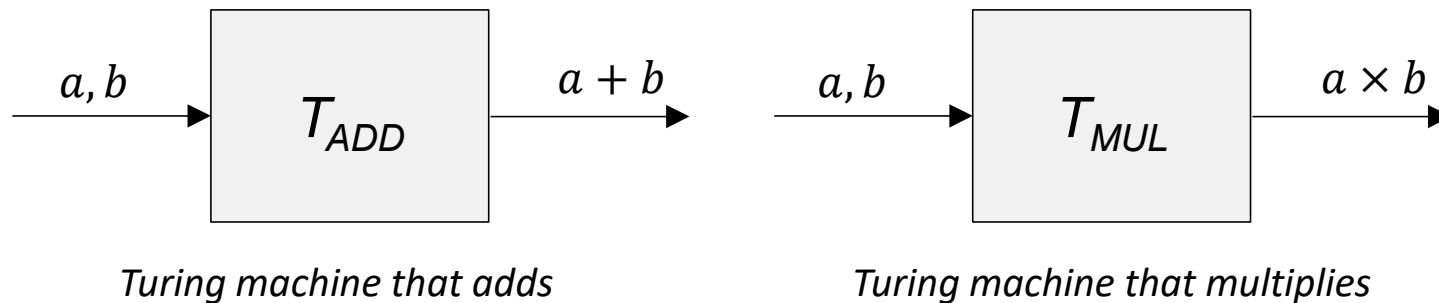


## ❖ Mathematical model of a device that can perform any computation

- proposed by Alan Turing in 1937 → 안 나옴 듯
- The Turing machine mathematically models a machine that mechanically operates on a tape. On this tape are symbols, which the machine can read and write, one at a time, using a tape head. Operation is fully determined by a finite set of elementary instruction. While he gave a mathematical description of this kind of machine, but did not actually build one

## ❖ Every computation can be performed by some Turing machine.

## ❖ Examples / Black Box Model



- The operation to be performed is described in the box
- It provides no information as to exactly how the operation is performed

# Universal Turing Machine

- ❖ A machine that can simulate all Turing machines ([Wikipedia](https://en.wikipedia.org/wiki/Universal_Turing_machine))
  - This is also a Turing machine
- ❖ Inputs : data + a description of computation (other TMs)



*Universal Turing Machine*

- ❖ Universal Turing Machine is **Programmable** !!!  $\rightarrow$  So is Computer !!!
  - A computer can emulate a Universal Turing Machine
- ❖ *A computer is a universal computing device.*

## Alan Turing (출처 Wikipedia)



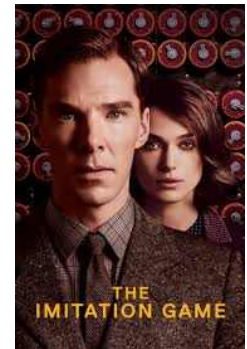
### ❖ 생애

- 1912년 출생, 1931년 케임브리지 대학 수학사 학위 취득, 1936년 미국 프린스턴 대학에서 박사 학위 취득.
- 1937년 튜링 기계 개념을 도입한 논문 "계산 가능한 수와 결정할 문제의 응용"을 발표, 1938년 영국 정부 암호학교 부임, 1939년 독일군의 에니그마 암호 해독기 개발
- 동성애자 혐의로 체포된 후 1954년 독이 든 사과를 먹고 죽음.

- ❖ 튜링은 수학, 암호학, 생물학 등 많은 분야에서 다양한 연구 활동을 했지만 특히 컴퓨터 과학 분야에 끼친 영향이 크기 때문에 컴퓨터 과학 및 전산학의 아버지라고 불린다. 그가 구상한 튜링 기계의 무한히 긴 띠는 컴퓨터의 메모리에, 기호를 읽는 기계는 컴퓨터의 CPU에 비유할 수 있다. 또한 튜링 기계의 한 종류인 범용 튜링 기계는 프로그램을 내장해서 작동하는 현대의 컴퓨터를 많이 닮아 있다. 미국컴퓨터학회 ACM에서는 튜링의 공로를 기리기 위하여 1966년부터 매년 컴퓨터 과학에 중요한 업적을 남긴 사람들에게 주는 튜링상을 제정하였다. 현재 튜링상은 컴퓨터 과학 분야의 노벨상이라고도 불린다.



애플 컴퓨터의 로고인 '한 입 베어먹은 사과'는 튜링을 연상시키지만, 애플 컴퓨터가 로고를 만들 때 튜링을 염두에 두고 만들었는지는 확실하지 않다. 현재 애플 컴퓨터에서는 로고의 모델이 뉴턴의 사과라고 주장한다.

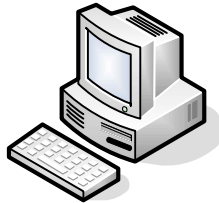


# 컴퓨터 시스템

컴퓨터 시스템은 사용자 요구에 따른 **임의의 다양한 계산을** 수행할 수 있는 장치로 개발되었다.

## 컴퓨터 시스템

H/W



S/W

- 컴퓨터 H/W로 수행할 수 있는 계산은 **명령어 (Instruction)**라 불리는 **한정된 기초 연산**에 불과하다.
- H/W만으로는 사용자가 요구하는 **임의의 다양한 계산을 수행할 수 없다.**

- **명령어를 조합하여 순차적으로 실행**하면 사용자가 요구하는 임의의 다양한 계산을 실현할 수 있다.
- 컴퓨터 시스템에서 S/W란 특정 계산을 위해 수행하는 명령어 조합과 관련 데이터 모음이다.

컴퓨터 시스템은 하드웨어와 소프트웨어로 이루어진다

양기



# Operation and Instruction

❖ Operation(연산) : **Operator**(연산자)와 **Operand**(피연산자)로 구성

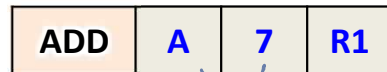


❖ Instruction(명령어) : **Opcode**(명령코드)와 **Operand**(피연산자)로 구성

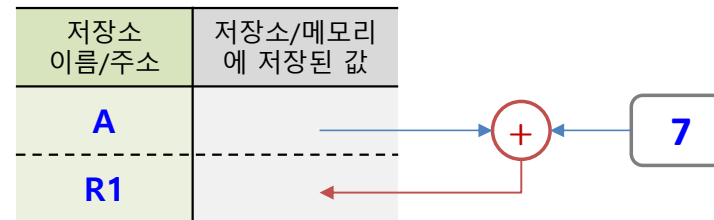


▪ 컴퓨터 명령어의 피연산자는 연산에 쓰이는 값(Literal) 또는 저장소/메모리(Memory)

✓ Example) 저장소 A에 저장된 값과 7을 더한 것을 저장소 R1에 저장하는 명령



- 많은 저장소/메모리 중 특정 저장소/메모리를 식별하기 위해 이름이나 주소를 부여
  - 건물이나 방의 번호/이름 등과 유사
- 저장소 A에 저장된 값에 따라 결과 값이 달라짐

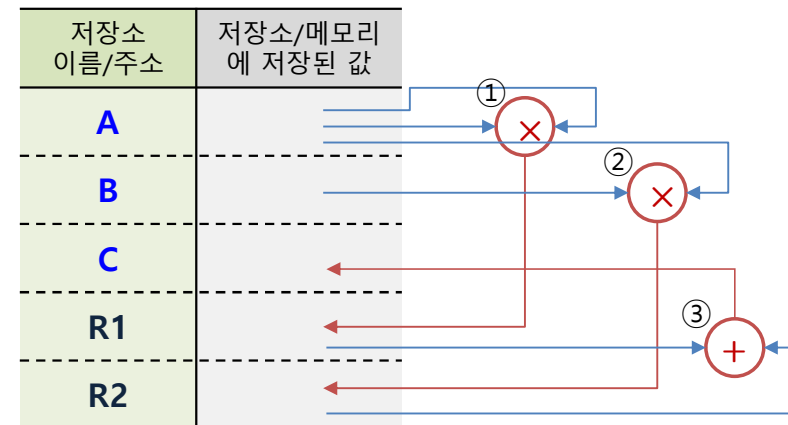


## Sequence of Instructions

### ❖ 명령어 조합을 통한 복잡한 계산 수행 예

- 저장소 A, B의 값에 대해 아래와 같은 계산을 수행하고 그 결과를 저장소 C에 저장하고자 한다.
- $A \times A + A \times B \rightarrow C$
- ✓ 그런데 대상 컴퓨터에는 이 계산을 바로 수행할 수 있는 연산 기능은 없다.
- ✓ 대상 컴퓨터는 앞 장에서 소개한 ADD, MUL 두 가지 명령어를 지원한다.
- ✓ 대상 컴퓨터에는 저장소 A, B, C 외에 R1, R2라는 추가 저장소를 임의로 사용할 수 있다고 가정한다.
- 아래와 같은 **Sequence of Instructions** (명령어 순차 조합)의 실행을 통해 원하는 (복잡한) 계산을 수행할 수 있다.

①	MUL	A	A	R1
②	MUL	A	B	R2
③	ADD	R1	R2	C

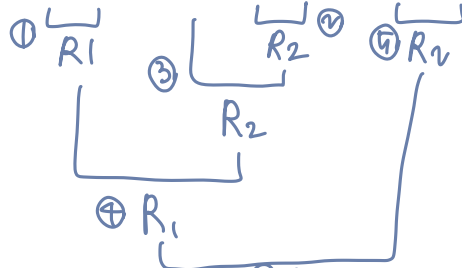


## Sequence of Instructions 예제 풀이

? 사용자가 원하는 계산이 다음과 같을 때  
이를 실현하는 명령어 순차 조합을 작성하라.

- 이전 예와 같이 추가 저장소 R1, R2를 사용할 수 있다.

■  $A \times A + 2 \times A \times B + B \times B \rightarrow C$



나랑 다르?

! 계산 수행을 위해 몇 개의 명령어를 사용하였는가?

- 동일한 기능을 더 적은 수의 명령어를 사용하여 실현하면  
보다 빠르게 목적을 달성
- 알고리즘(Algorithm)의 필요성

	Opcode	Operands		
①	Mul	A	A	R1
②	"	A	B	R2
③	"	2	AB	R2
④	"	A <sup>2</sup>	2AB	R1
⑤	"	B	B	R2
⑥	"	2A <sup>2</sup> B	B <sup>2</sup>	C
⑦				
⑧				
⑨				
⑩				
⑪				
⑫				
⑬				
⑭				

## H/W Instruction and S/W



*Lego Blocks → Instructions*

**H/W**



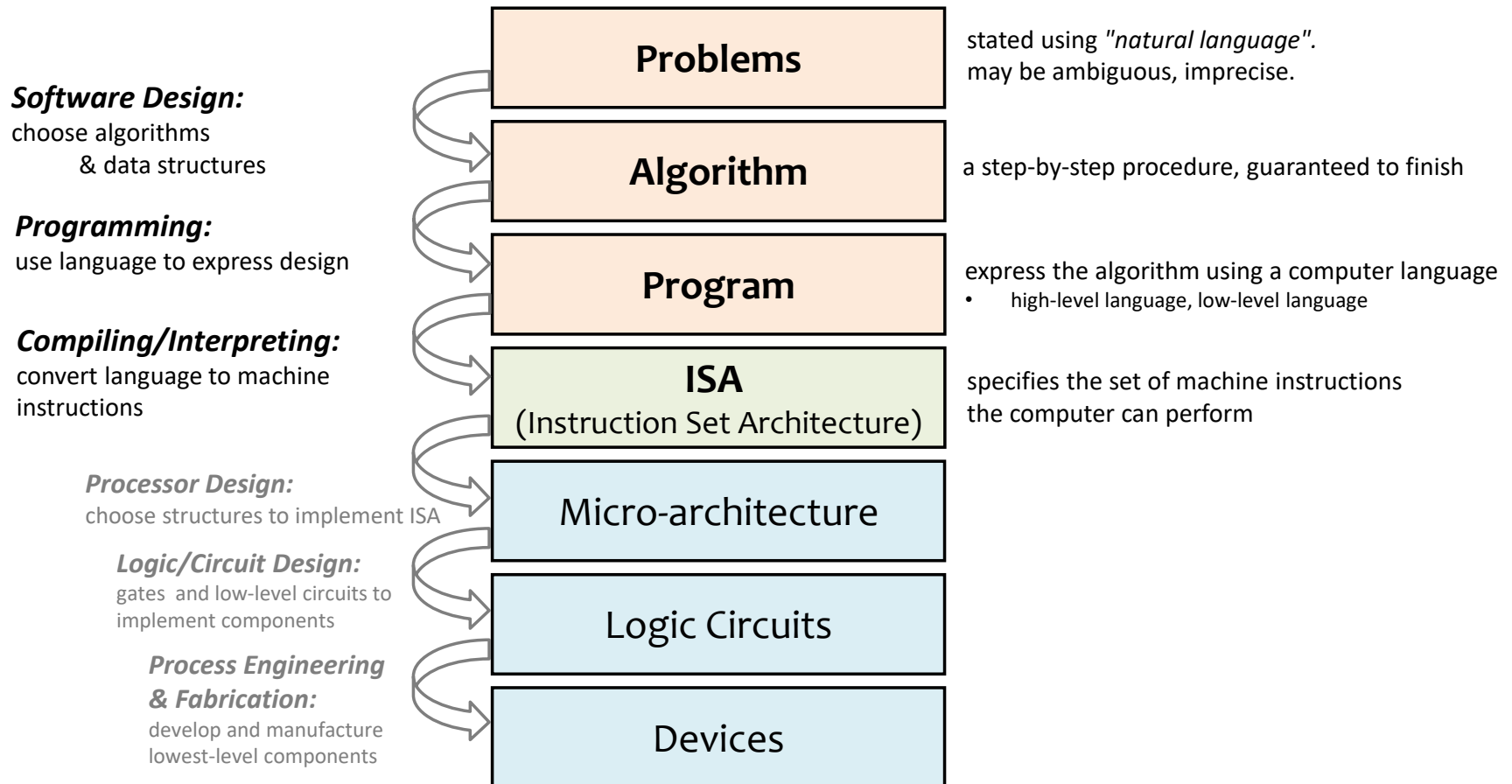
*Ideas → Sequence of Instructions/Program*

**S/W**

# How do we get the electrons to do the work ?

- How do we solve a problem using a computer ?

A systematic sequence of *transformations* between *layers of abstraction*.



# Properties of an Algorithm

❖ *An algorithm is a step-by-step procedure that is guaranteed to terminate, such that each step is precisely stated and can be carried out by the computer.*

## ❖ Definiteness :

- Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
- Ex) In a recipe for pancakes : "stir until lumpy". Does it meet "definiteness" ?

## ❖ Effective Computability :

- All operations to be performed must be sufficiently basic that they can be done exactly and in finite length
- Ex) "Take the largest prime number". Does it meet "effective computability" ?

## ❖ Finiteness :

- The algorithm must always terminate after a finite number of steps.
- Ex) "Divide x by 2, repeat it until x has a value of 1". Does it meet "finiteness" ?

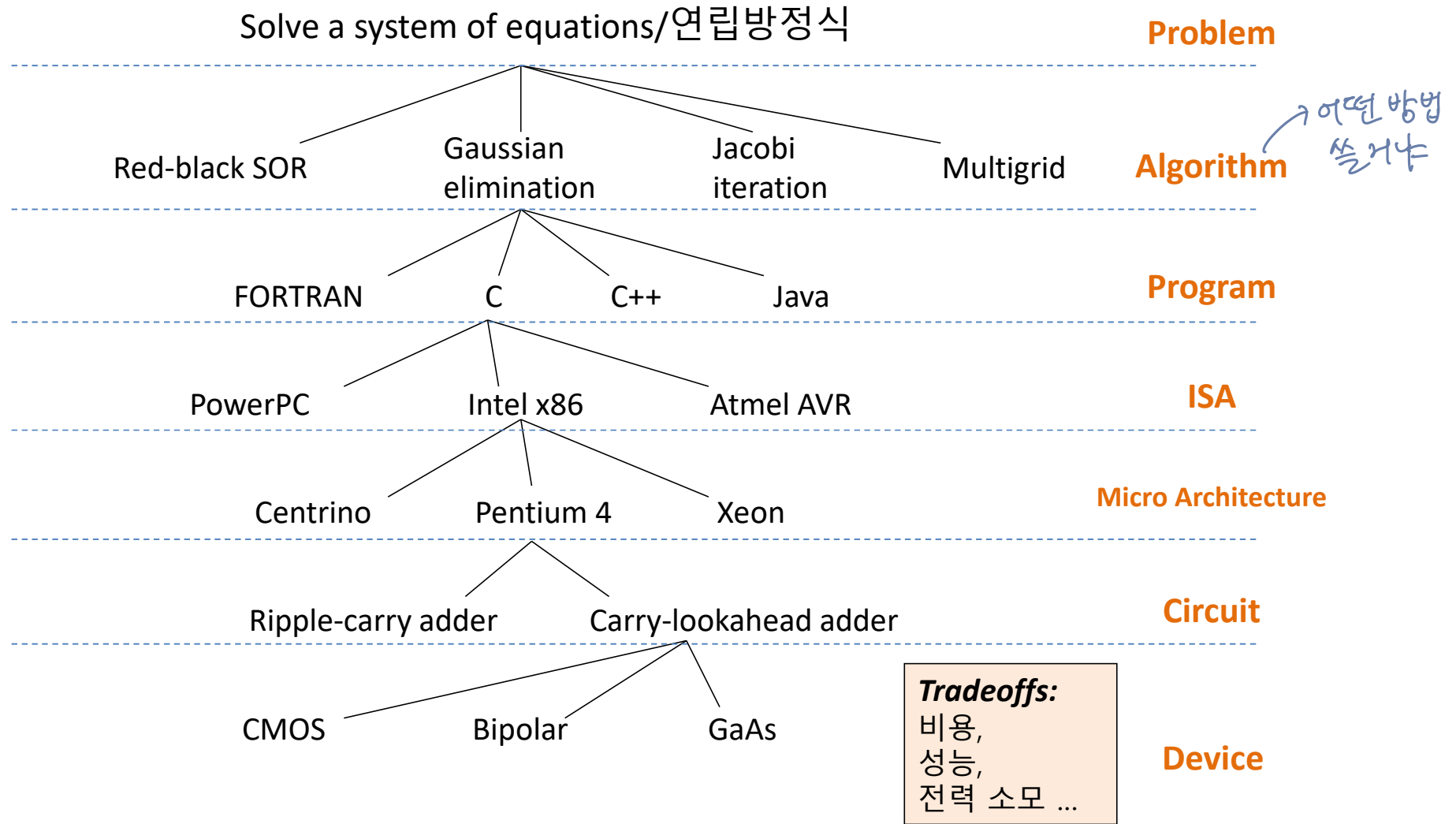
## ❖ Input

- An algorithm has zero or more inputs, taken from a specified set of objects.

## ❖ Output

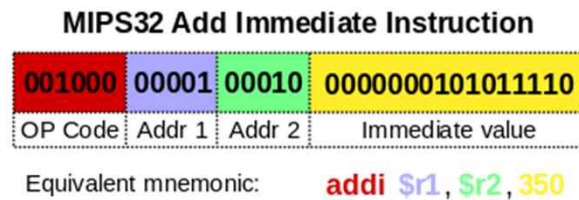
- An algorithm has one or more outputs, which have a specified relation to the inputs

# Many Choices at Each Level



## ❖ ISA (Instruction Set Architecture)

- 이전 슬라이드에서 소개한 ADD, MUL 등은 어셈블리 언어(Assembly Language)로 표현한 명령
- CPU가 직접 해독하고 실행할 수 있는 명령어 0과 1의 이진 값으로 표현되는 기계어 명령어
  - 다음은 예)



- CPU가 직접 해독하고 실행할 수 있는 기계 명령어들의 집합 그리고 그 기반 구조 및 체계 → **ISA**

## ❖ ISA 분류

- ISA는 구조적 특성에 따라 <sup>인텔</sup>CISC (Complex Instruction Set Computer), <sup>ARM</sup>RISC (Reduced Instruction Set Computer) 등으로 분류
- CISC 구조의 대표 CPU/ISA로 Intel의 x86 Architecture (Instructions) – PC/Notebook 등에 널리 쓰임
- RISC 구조의 대표 CPU/ISA로 ARM의 ARM Architecture – 다양한 Embedded 장치 및 모바일 기기에 널리 쓰임



# C 언어 소개와 컴파일의 이해

프로그래밍 언어 분류

C 언어 소개

Compiled Language vs. Interpreted Language

C 언어 컴파일과 Python 언어 인터프리터 비교하기

Von Neumann Architecture

# 프로그래밍 언어와 분류


## ❖ 프로그래밍 언어 - 프로그램을 작성하기 위한 언어

- 사람이 컴퓨터에게 시키고 싶은 내용을 표현하기 위한 표기법
- 인간 친화성 수준에 따라 **기계어**(Machine Code/Language), **어셈블리어**(Assembly Language), **고급 언어**(High-Level Language)로 분류

## ❖ 기계어

- 이진수 코드로 CPU 마다 고유의 기계어 → ISA

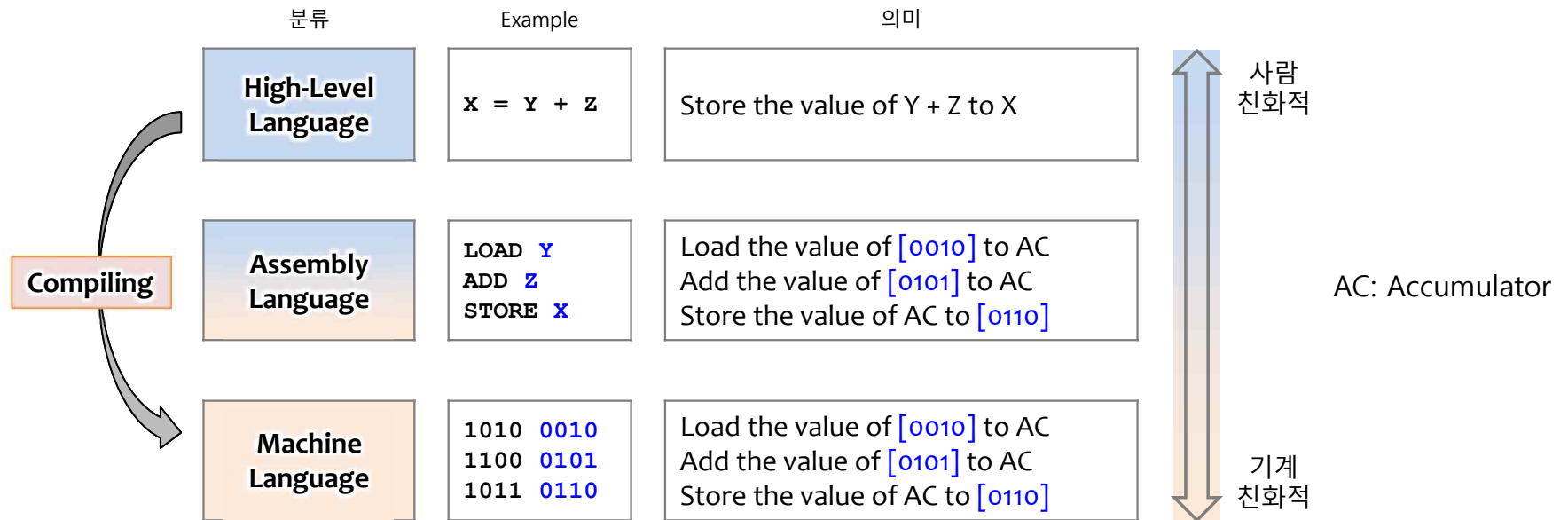
## ❖ 어셈블리어

- 
- 기계어 이진수 코드를 사람이 이해하기 쉬운 기호화 코드(mnemonics)로 대치한 것
  - Symbolic Machine Code로도 불림
    - 참고) 많은 경우 기계어 코드와 어셈블리어 기호 사이에 1:1의 대응 관계에 있지만 그렇지 않은 경우도 있음.

## ❖ 고급 언어

- 영어와 비슷한 구문으로 표현되며, 사람이 읽고 쓰기가 보다 쉽다
- 기계어에 독립적이어서 프로그래머가 기계의 세부사항(ISA)을 알지 못해도 프로그램을 작성할 수 있다.
- C/C++, Python, Java 등이 고급 언어의 예
- 대비의 관점에서 어셈블리어나 기계어를 저급 언어(Low-Level Language)라고도 함

# 기계어, 어셈블리어, 고급 언어



- 보통 프로그래머는 고급 언어로 프로그램의 **소스 코드(Source Code)**를 작성하고 **컴파일(Compiling)**이라는 과정을 통해 기계가 해독하여 처리할 수 있는 기계어 형식의 **이진실행파일(Binary Executable)**로 변환한다.
- 이진실행파일 형태의 프로그램을 운영체제를 이용하여 적재하고 실행한다.

# 기계어와 이식성

## ❖ Portability (이식성) → 다른 기계어에서 잘 동작하는 성질

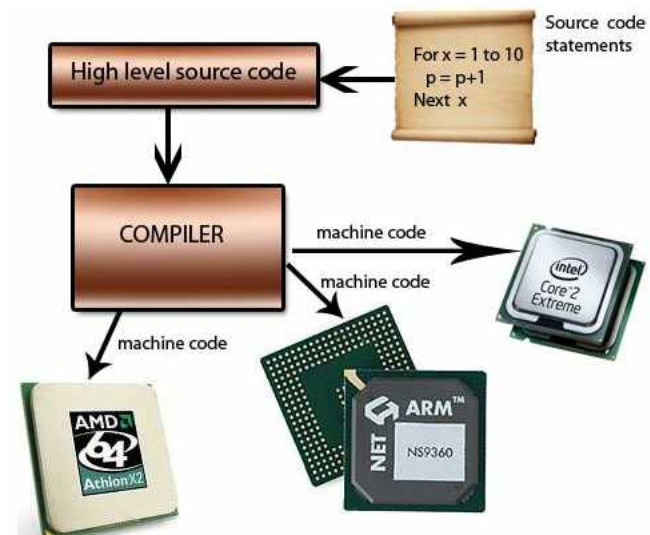
- 프로그램을 다른 기계/플랫폼에서 활용하기 위해 필요한 수정이나 변경의 수준
- *High Portability* : 수정, 변경이 거의 없이 프로그램을 사용할 수 있는 것

## ❖ 기계어로 작성된 프로그램은 이식성이 매우 낮음

- CPU 마다 다른 기계어(ISA)를 사용하기 때문

## ❖ 고급 언어의 경우 상대적으로 이식성이 높음

- 단 다른 CPU/ISA에서 사용하기 위해서는  
소스 코드를 다시 컴파일 하여  
해당 CPU/ISA에 적합한 이진실행파일을  
생성하여야 함



# c 언어 소개

## ❖ Dennis Ritchie와 Ken Thompson이 개발

- 1969년 ~ 1973년 사이 Bell Labs 근무 당시 UNIX 운영체제 개발에 사용됨
- 미국 및 국제 표준으로 제정
  - ANSI (American National Standard Institute) C (1989년)



[Ken Thompson](#) (left) with [Dennis Ritchie](#) (right, the inventor of the C programming language)  
Source - Wikipedia

## ❖ c 언어의 특징

- 고급 언어의 특성(쉬움)과 저급 언어의 강점 (빠르고 효율적)을 겸비

특징	장점
<u>구조화(Structured) 프로그래밍</u> 지원	복잡한 문제를 잘 정의된 여러 개의 작은 함수로 나누어 구성, 해결할 수 있다. 블록(Block), 함수(Sub-Routine) 등의 개념을 통해 프로그램 소스 코드의 구조를 이해하기 쉽게 작성하고 읽을 수 있다
이식성 (Portability)	다양한 컴퓨터 플랫폼에서 사용할 수 있다 어셈블리 언어나 기계어에 비교할 때 큰 장점
효율성 (Efficiency)	적은 메모리를 이용하여 빠르게 수행 가능한 프로그램을 작성할 수 있음
다양한 연산자	프로그램을 간결하고 쉽게 작성할 수 있음. 예) $x = x + 1;$ → $x++;$
동적 메모리 (Dynamic Memory) 관리	동적인 메모리 할당과 해제를 통해 임베디드 시스템과 같이 메모리 자원이 제한된 환경에서도 구동할 수 있는 프로그램을 작성할 수 있음

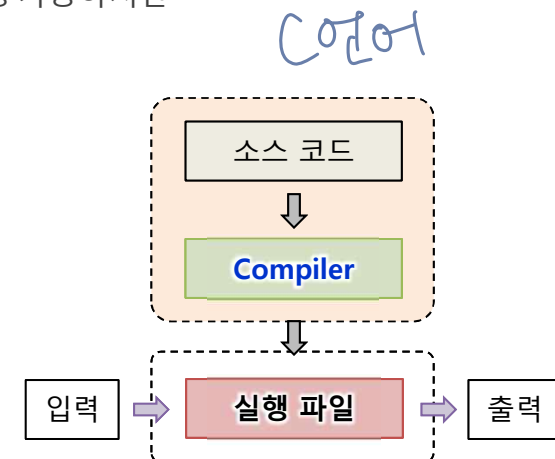
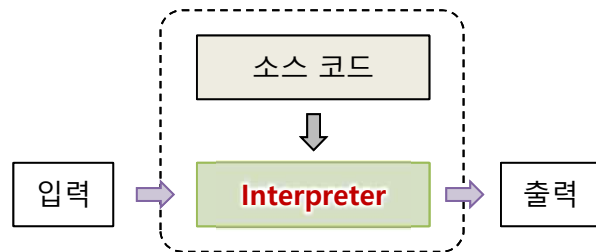
→ 메모리 안전 사용 / 관리

# Compiled Language vs. Interpreted Language

비교항목	Compiled Language	Interpreted Language
대표 언어	C/C++, Fortran, Cobol	Python, Java, Basic
실행/목적 파일	생성	생성 안 함
실행 속도	빠름	상대적으로 느림 <i>큰 번역 과정 필요</i>
이식성 (Portability)	상대적으로 낮음. ISA가 다른 컴퓨터에서는 다시 컴파일 해야 함	높음 인터프리터만 있으면 ISA가 다른 컴퓨터 에서도 실행 가능

## ❖ 실행/목적 파일의 생성 여부가 중요한 차이

- Compiled Language는 **전체 소스 코드**를 이진 기계어 코드로 번역하고 그것을 묶어 **이진 실행 파일을 생성함**
- Interpreted Language의 경우 **소스 코드의 일부 (일반적으로 한 문장씩)**를 이진 기계어 코드로 번역하고 그 결과를 바로 실행하게 함.  
즉 별도의 **이진 실행 파일을 생성하지 않음**.
- Compiled Language의 경우 실행/목적 파일이 있으면 소스코드와 Compiler가 없어도 실행 가능하지만  
Interpreted Language의 경우 실행을 위해 소스 코드와 Interpreter가 필요
  - 참고) Python 소스를 interpretation하지 않고 Compile을 통해 실행 파일을 만들 수도 있다.  
이 경우 Python을 Compiled Language로 분류할 수 있다.



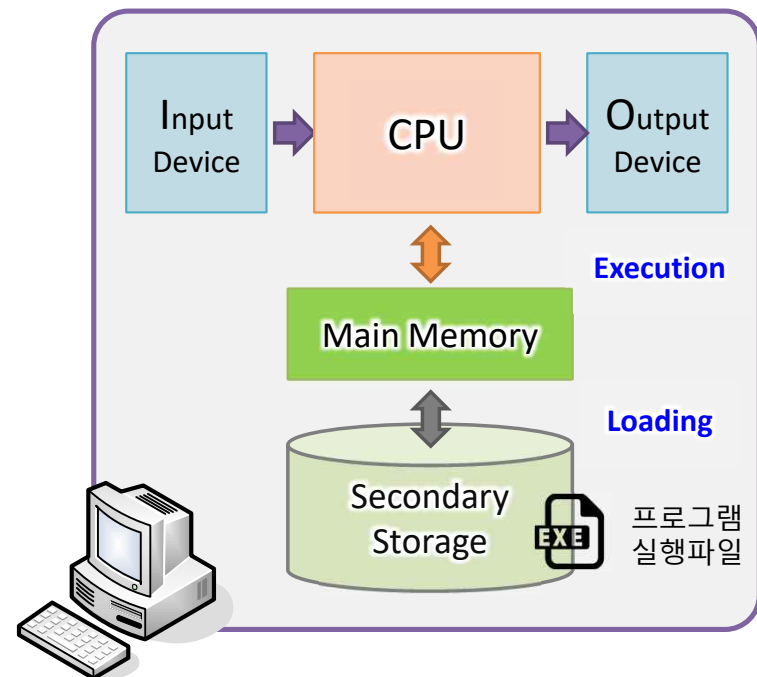
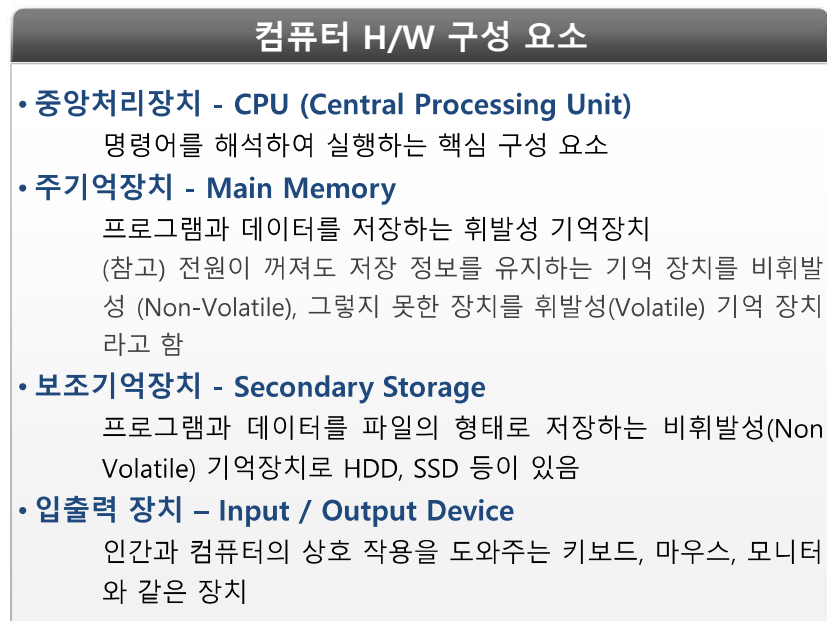
# 컴퓨터 프로그램과 실행

## ❖ 컴퓨터 프로그램(Program)이란?

- **Sequence of instructions and related data !!!**
- 명령어는 기계가 직접 해독하고 실행할 수 있는 이진 값으로 표현되는 기계어 (Machine Code)
- 실행 파일(Executable File)의 형태로 보조기억장치에 저장/설치됨

## ❖ 프로그램의 실행 과정

- 보조기억장치에 저장된 프로그램 실행 파일을 Main Memory로 **적재 (Loading)**하고
- CPU는 Main Memory로부터 프로그램의 명령어들을 순차적으로 하나씩 읽어 들여 해석하고 수행하는 방식으로 프로그램을 **실행(Execution)**
  - 참고) 프로그램 적재/실행을 수행하는 S/W → 운영체제(OS : Operating System)

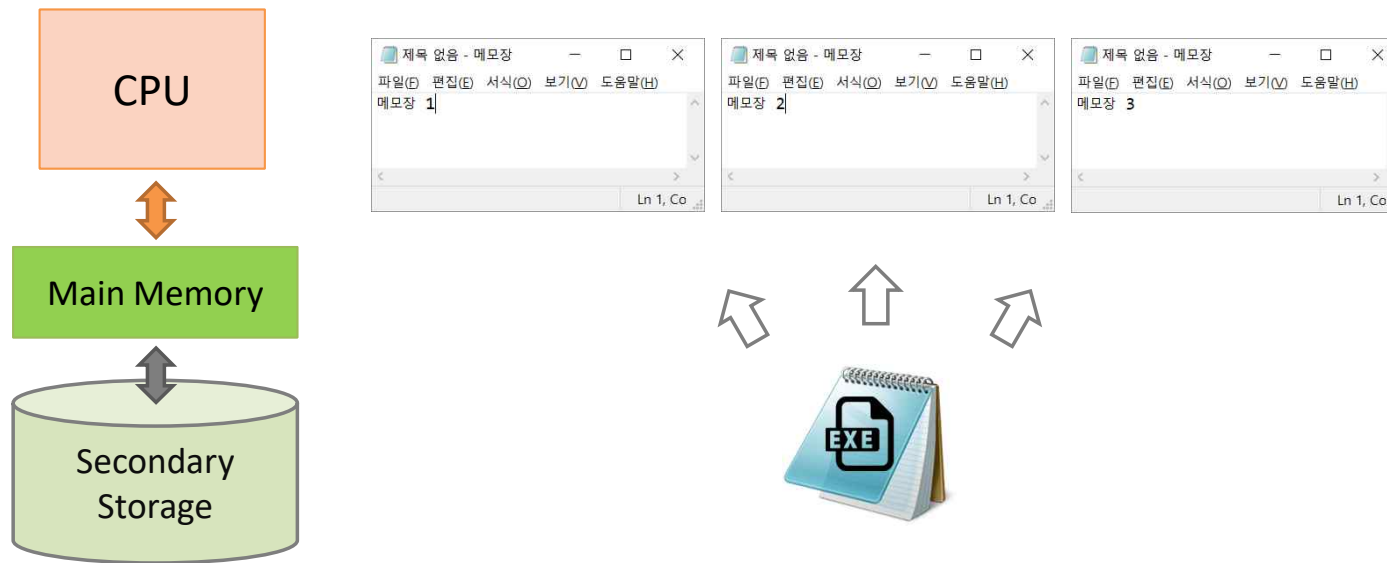


## ❖ 프로그램과 프로세스의 차이 : Program vs. Process

- A **program** is an **executable file residing on the disk** (secondary storage). It is read into the primary memory and executed by the kernel.
- An **executing instance of a program** is called a **process**

## ❖ 다음을 수행하라.

- 컴퓨터에서 메모장 프로그램의 실행파일을 찾아라. 확장자 포함 파일 이름이 무엇인가?
- 프로그램을 실행하여 3개의 메모장 프로세스를 생성하라.





# Bit, Byte, Hexa

## ❖ 컴퓨터에서 정보는 2진수로 표현

- 예) 0100, 11010001

## ❖ Bit – Binary Digit

- 컴퓨터에서 정보를 표현하는 기본 단위
- 1 bit는 0 또는 1을 나타낼 수 있음
- $n$  bits는  $2^n$  개의 경우의 수를 표현할 수 있음

## ❖ 1 Byte = 8 bits

- 알파벳 문자 표현을 위해 8 bits가 필요했던 것으로부터 유래
- 1 Byte는 256가지의 경우의 수 표현 가능

28 정보 표현 가능

Value	Prefix	Symbol	Value	Binary Prefix
$10^3=1000$	Kilo	K	$2^{10}=1024$	Kibit
$10^6$	Mega	M	$2^{20}$	Mibit
$10^9$	Giga	G	$2^{30}$	Gibit
$10^{12}$	Tera	T	$2^{40}$	Tibit
$10^{15}$	Peta	P	$2^{50}$	Pibit
$10^{18}$	Exa	E	$2^{60}$	Eibit
$10^{21}$	Zetta	Z	$2^{70}$	Zibit
$10^{24}$	Yota	Y	$2^{80}$	Yibit

## ❖ Hexa

- 16진수(Hexa Decimal) 체계
- 0, 1, ..., 9, A(10), B(11), C(12), D(13), E(14), F(15)
- 긴 2진수를 짧게 표현 가능, 하나의 숫자가 4 bits에 대응
- (Hexa임을 표시하기 위해 앞에 0x를 덧붙이는) 경우가 많음
- 예) 001010101100 → 0x2AC

4비트로 쓸수있는 정보 : 16  
→ 16개

2 A C

# Memory Address

## ❖ Memory

- $m$  bits를 저장할 수 있는 다수의 Memory Cell이 연속으로 이어져 있는 것
- 대부분의 현대 컴퓨터는  $m = 8$ , 즉 Memory Cell이 1 Byte를 저장

## ❖ (Memory Address)의 필요성

- 연속된 Memory Cell 중 어떤 Cell에서 읽거나 쓸지를 표현할 수 있어야 함 → Address
- 수 많은 방 중 어떤 방?



## ❖ Memory Address

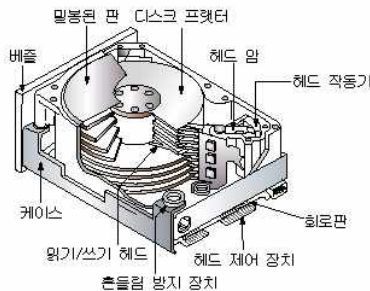
- 컴퓨터에서는 주로 2진수 또는 Hexa로 표현
- Memory Address에 쓰이는 bits 수 →  $k$ ,  $2^k$ 개의 Memory Cell 구별
- $k$  값은 전체 메모리 공간의 크기와 관련
- 예)  $k = 32, m = 8$  인 메모리 공간의 크기는 4GiBytes

Address Memory Cells  
 $k = 6$   $m = 8$

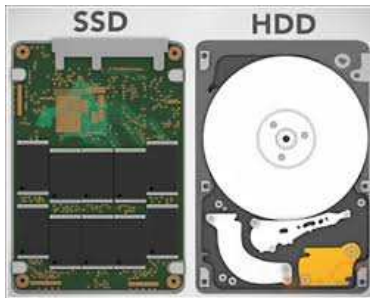
000000	10011010
000001	00000000
000010	10011010
000011	00000000
...	...
111100	10011010
111101	00000000
111110	10011010
111111	00000000

## 보조 자료

### ➤ HD (Hard Disk) / HDD (Hard Disk Drive)



### ➤ SSD (Solid State Drive)



하드디스크는 자기 물질을 표면에 얇게 입힌 금속 재질의 원형 판(disk, platter)으로 디지털 정보를 쓰고 읽을 수 있는 데이터 저장 장치다. 하드디스크는 컴퓨터 전원이 꺼져도 정보가 유지되는 비휘발성(Non-Volatile) 기억 장치이다.

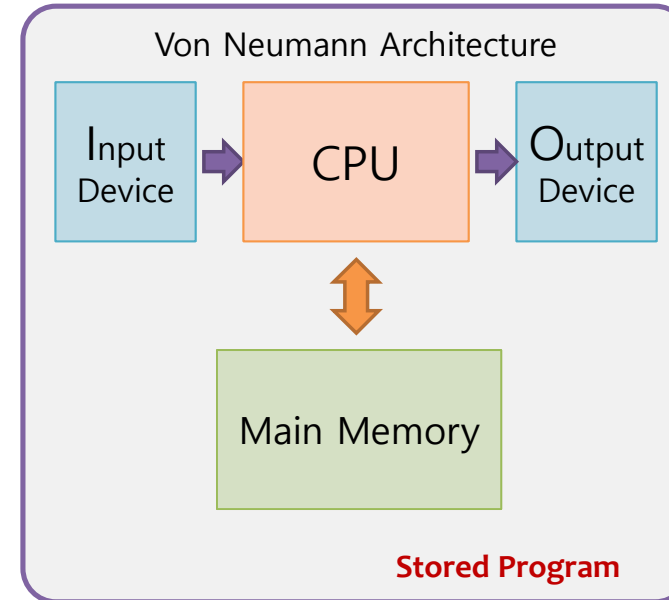
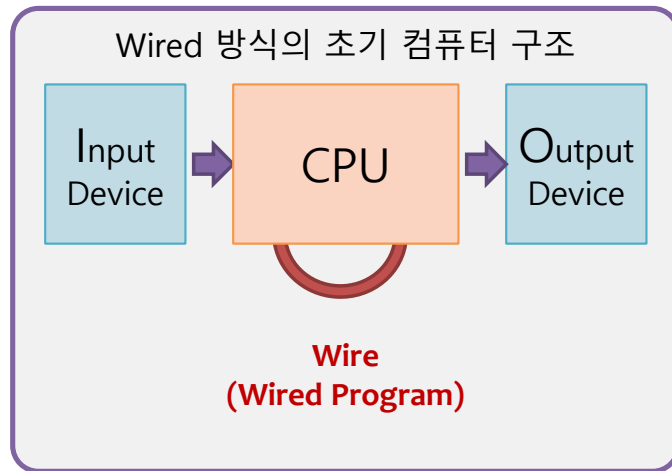
하드디스크는 그 표면에 정보를 쓰거나 읽을 때 쓰이는 헤드와 쌍을 이룬다. 하드디스크 드라이브는 하드디스크를 빠르게 회전 시키고 헤드의 위치를 조절하면서 임의의 위치에 데이터를 읽고 쓸 수 있는 기계 장치이다.

하드디스크 크기로는 5.25인치(13.34cm), 3.5인치(8.9cm), 2.5인치(6.4cm), 1.8인치 (4.6cm) 등이 있으며 분당 회전 수, 즉 RPM은 5400, 7200 등이 있다. 현재 데스크탑 PC에는 3.5인치, 노트북에는 2.5인치 HDD가 주로 쓰인다.

플래쉬 메모리 반도체를 저장 매체로 사용하는 데이터 저장 장치이다. 반도체이나 일반적인 메모리 반도체와 달리 전원이 꺼져도 정보가 유지되는 비휘발성(Non-Volatile) 기억 장치이다.

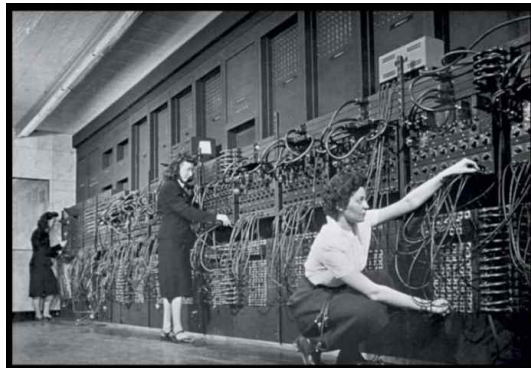
순수 전자식으로 작동하므로 기계식인 HDD의 문제인 긴 탐색 시간, 반응 시간, 기계적 지연, 실패율, 소음을 크게 줄여 준다. 또한 가볍고 전원도 적게 사용한다. 예전에는 데이터 접근 시간이 아닌 연속적인 읽기와 쓰기에 대해 HDD 보다 느린 경우가 많았지만 최신 기술이 적용된 SSD의 경우 연속적인 읽기 쓰기에서도 HDD보다 빠르다. SSD의 최대 단점은 비싼 가격이었으나 최근에는 가격이 빠르게 하락하고 있다.

# Von Neumann Architecture - History



1943: ENIAC

- 최초의 범용 전자 컴퓨터
- **Hard Wired Program - 다이얼과 스위치 조작**



1944: Beginnings of EDVAC

- **프로그램을 메모리에 저장**할 수 있는 기능 개선 이루어짐

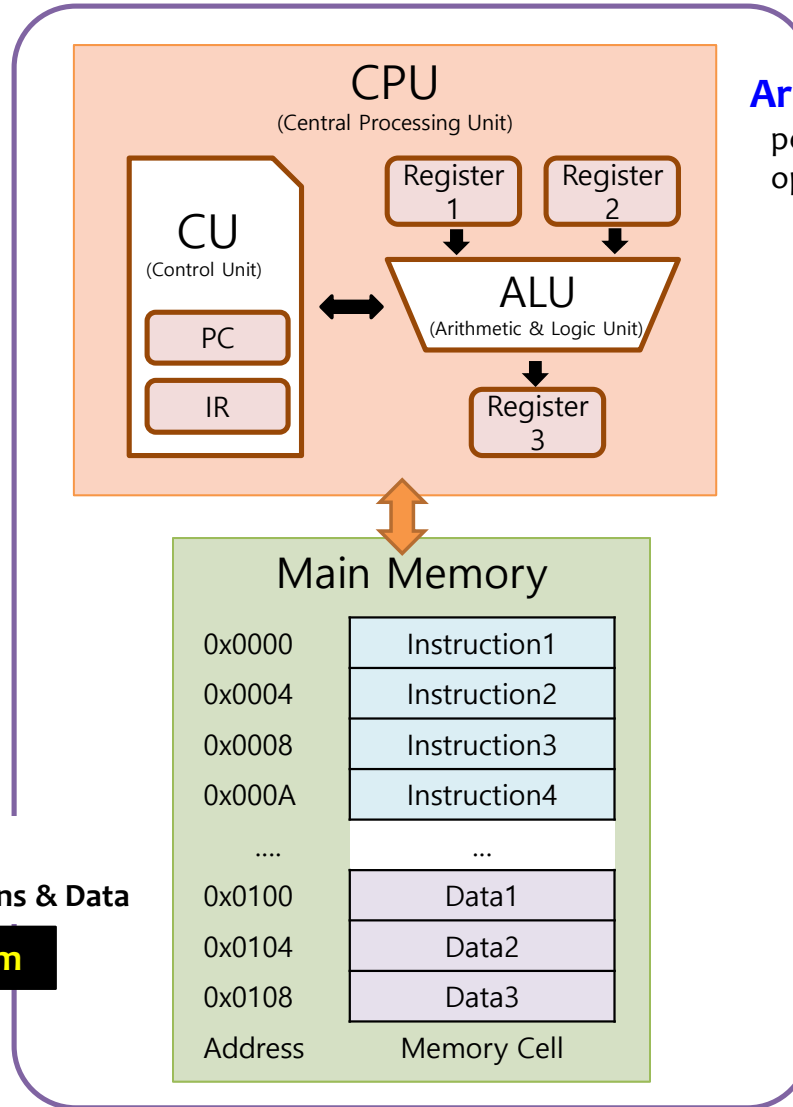
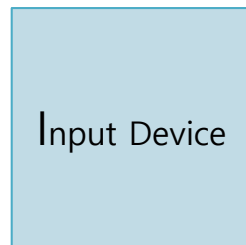
1945: John von Neumann

- “The First Draft of a Report on EDVAC”이라는 문서를 통해 **“Stored Program”** 개념 발표
- Draft에서 설명한 기본 구조 → Von Neumann Machine (or Model)로 불림

# Von Neumann Architecture

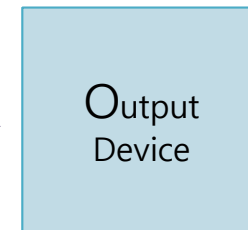
## Control Unit :

interpreting **Instructions**



## Arithmetic & Logic Unit :

performing arithmetic and logic operations



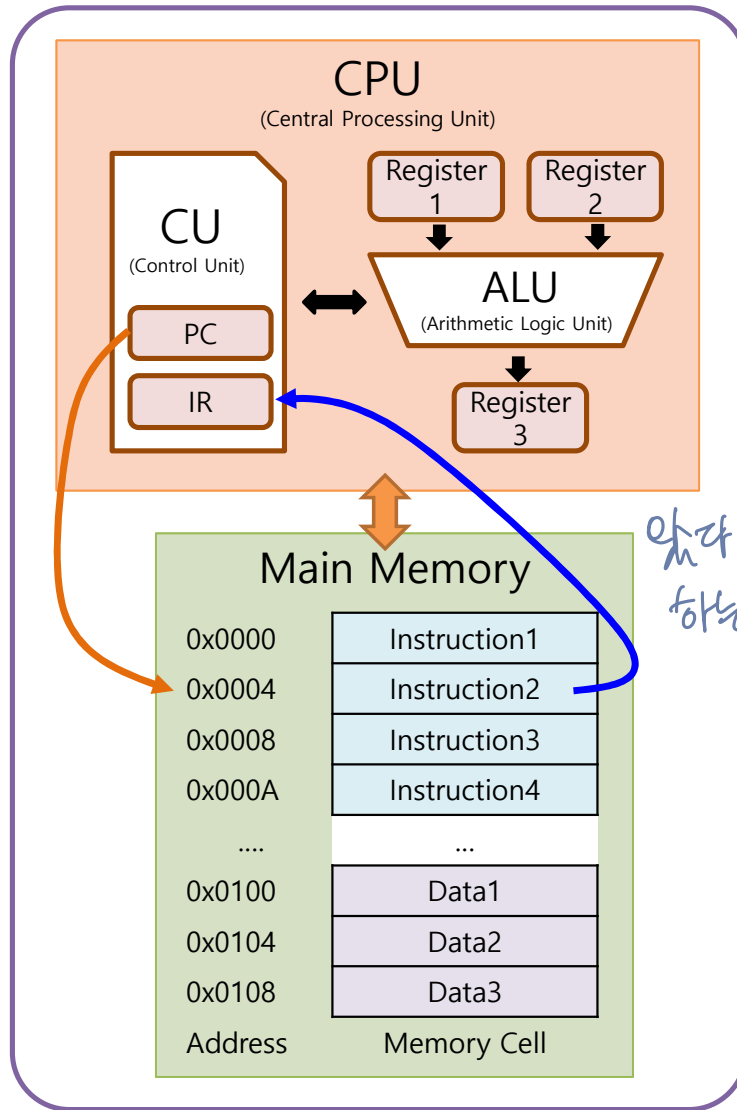
## Memory :

Contains **Instructions & Data**

**Stored Program**

# Von Neumann Architecture

PC : Program Counter  
IR : Instruction Register



## ❖ CPU : ALU + CU + Registers

## ❖ ALU (Arithmetic and Logic Unit)

- 연산을 수행하는 Unit

## ❖ Register

- ALU 연산 작업 등에 쓰이는 CPU 내의 작은 용량의 고속 Memory

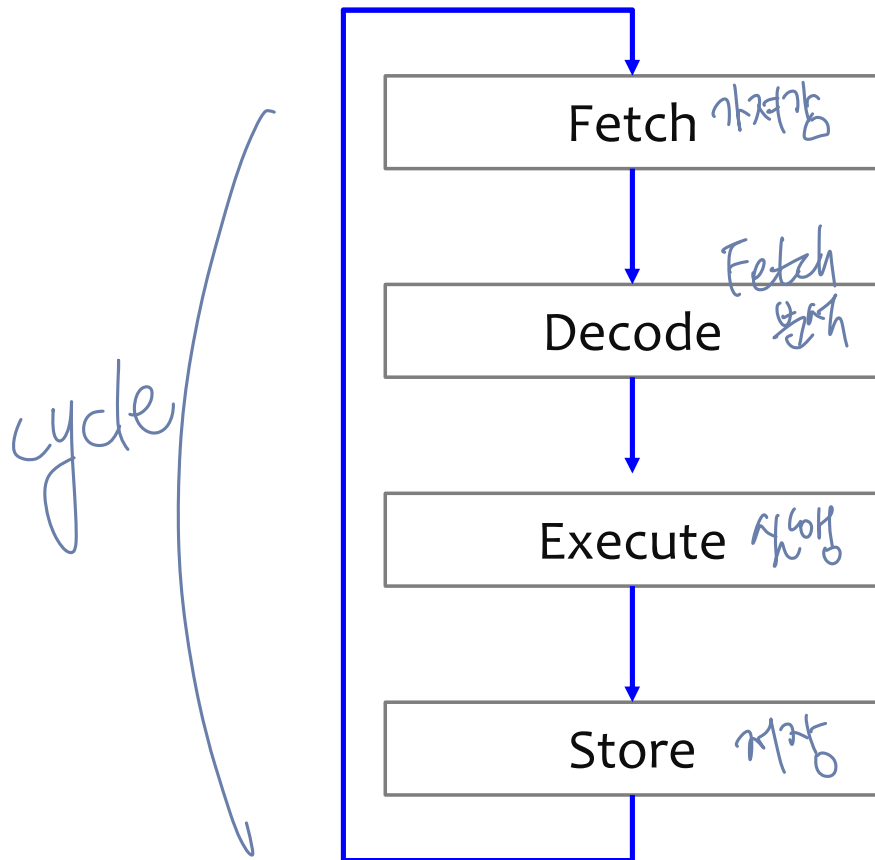
## ❖ CU (Control Unit)

- Instruction을 Main Memory에서 읽어와 ALU와 Register를 활용하여 명령을 수행하는 과정을 제어하는 Unit
- 현재 읽어 들여 수행할 Instruction이 저장된 메모리의 주소 값을 가지는 PC (Program Counter)라고 불리는 Register가 존재함
  - PC 값을 이용해 수행할 Instruction을 IR(Instruction Register)라는 Register에 읽어옴
  - Instruction을 수행하면 일반적으로  $PC = PC + 1$ 의 값을 가짐, 즉 다음 Instruction이 저장된 주소를 가짐

앞다 갔다 하는 것 : 사이클 cycle

# Instruction Cycle

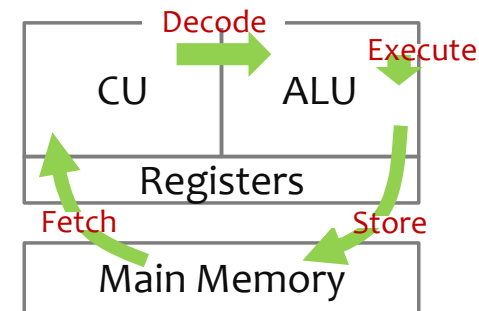
- ❖ The cycle which the CPU follows from boot-up until the computer has shut down in order to process instructions



The next instruction is fetched from the memory address that is currently stored in the PC(program counter) and stored into the IR(instruction register). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.

The encoded instruction present in the IR is interpreted by the CU.

The CU passes the decoded information as a sequence of control signals to the ALU to perform mathematical or logic operations



## Eclipse

- Debug
- Disassembly

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes a 'Registers' button. The left sidebar shows the 'Debug' view with a tree structure of the application and threads. The main editor area is split into three panes: 'main.c' (Source Code), 'Registers', and 'Disassembly'.

**Registers Panel:**

Name	Value	Description
<b>General Registers</b>		
eax	1	
ecx	4200704	
edx	128	
ebx	3375104	
esp	0x61ff00	
ebp	0x61ff28	
esi	4199104	
edi	4199104	
eip	0x40144e <main+14>	
eflags	[ IF ]	
cs	35	
--	42	

**Source Code Panel (main.c):**

```
2 * main.c
7
8 #include <stdio.h>
9
10 int main(void) {
11     int a = 1;
12     int b = 2;
13
14     a = a+b;
15
16     printf("a+b=%d\n", a);
17
18     return 0;
19 }
20
21
```

**Disassembly Panel:**

```
00401449: call 0x401980 <_main>
11     int a = 1;
0040144e: movl $0x1,0x1c(%esp)
12     int b = 2;
00401456: movl $0x2,0x18(%esp)
14     a = a+b;
0040145e: mov 0x18(%esp),%eax
00401462: add %eax,0x1c(%esp)
16     printf("a+b=%d\n", a);
00401466: mov 0x1c(%esp),%eax
0040146a: mov %eax,0x4(%esp)
0040146e: movl $0x40a064,(%esp)
00401475: call 0x408bc0 <printf>
18     return 0;
0040147a: mov $0x0,%eax
19 }
0040147f: leave
00401480: ret
00401481: nop
00401482: nop
00401483: nop
```

Blue arrows indicate the flow from the source code to the disassembly. A 'C Source Code' label points to the source code pane, and a 'Disassembly' label points to the disassembly pane.



## John von Neumann (출처 Wikipedia)



### ❖ 생애

- 1903년 헝가리 부다페스트에서 부유한 유대인 은행가의 장남으로 출생
- 1926년 23세의 나이로 부다페스트 대학 수학 박사 학위 취득,  
1930년까지 베를린 훔볼트 대학에서 강사 생활
- 1930년 프린스턴 고등연구소로 초청을 받아 미국으로 건너가 고등연구소 최초 4명의 교수진 중에 한 명이 됨. 이후 죽을 때까지 고등연구소의 수학 교수로 활동
- 2차 대전 기간 핵무기 개발을 위한 맨하탄 프로젝트 참여, 핵무기 개발에 지대한 기여, 최초의 핵폭파 실험을 직접 관찰한 소수의 과학자 중 한 사람, 이후 미 정부의 대륙간탄도탄(ICBM) 위원회에 참여, 반공 보수주의자로 적극적인 핵무기 옹호자, 균형 이론에 기초하여 그가 주장한 상호 확증 파괴, 즉 MAD(Mutually Assured Destruction)는 냉전 시기 미, 소의 핵무기 전략이었음.
- 1957년 방사능이 원인으로 추정되는 골수암에 걸려서 사망



### ❖ 지상 최강의 천재로 알려진 존 폰 노이만은 너무 머리가 좋아서 화성인, 악마의 두뇌를 가진 남자라고 불리웠음.

7개의 외국어를 모국어 같이 자유자재로 사용할 수 있었으며, 발음 또한 완벽하다고 전해짐. 역사상 가장 뛰어난 기억력을 가진 인물로 평가되는데 훈련이 아니라 타고난 능력. 양자 물리학, 함수 해석학, 집합 이론, 컴퓨터 과학, 경제학 등의 분야에서 다수의 중요한 공헌을 하였음. 게임 이론의 창시자. DNA/RNA 발견 이전에 그 존재를 예측한 Cellular Automata / Universal Constructor 개념의 창시자. 그의 천재성과 관련한 수많은 일화들이 있음. ([참고 1](#), [참고 2](#))

- EDVAC이 완성되어 시험을 하게되었다. 누군가 “오른쪽에서 4번째 자리수가 7인 가장 작은 2의 지수는 얼마인가?” 컴퓨터와 폰 노이만이 동시에 문제를 풀기 시작했고 폰 노이만이 먼저 풀어 승리하였다.