

“본 강의 동영상 및 자료는 대한민국 저작권법을 준수합니다. 본 강의 동영상 및 자료는 상명대학교 재학생들의 수업목적으로 제작·배포되는 것이므로, 수업목적으로 내려받은 강의 동영상 및 자료는 수업목적 이외에 다른 용도로 사용할 수 없으며, 다른 장소 및 타인에게 복제, 전송하여 공유할 수 없습니다. 이를 위반해서 발생하는 모든 법적 책임은 행위 주체인 본인에게 있습니다.”



# 피지컬 컴퓨팅

## Lec. 2. 컴퓨터 구조 (CPU, Memory, Bus)

Heenam Yoon

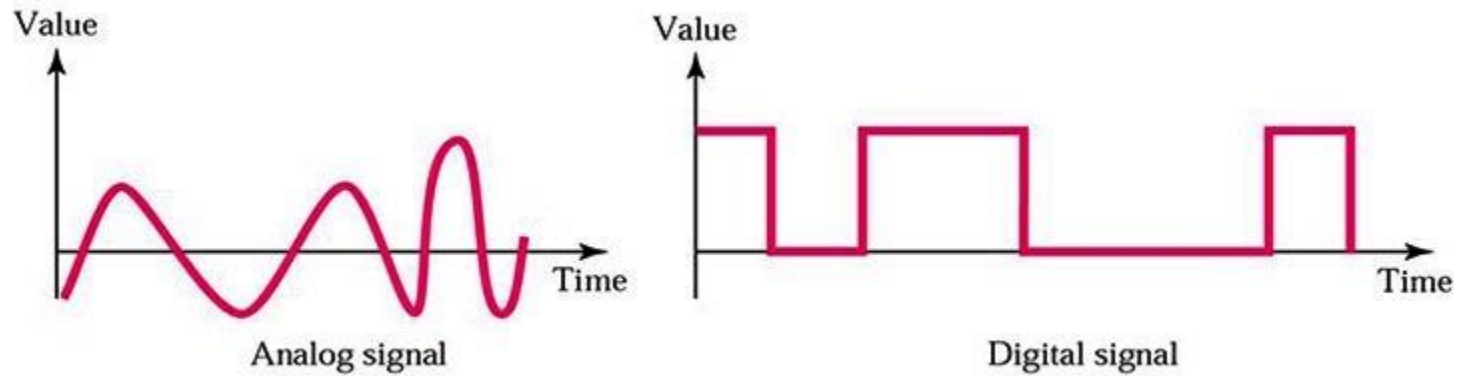
Department of  
Human-Centered Artificial Intelligence

E-mail) [h-yoon@smu.ac.kr](mailto:h-yoon@smu.ac.kr)  
Room) 0112



# 기초

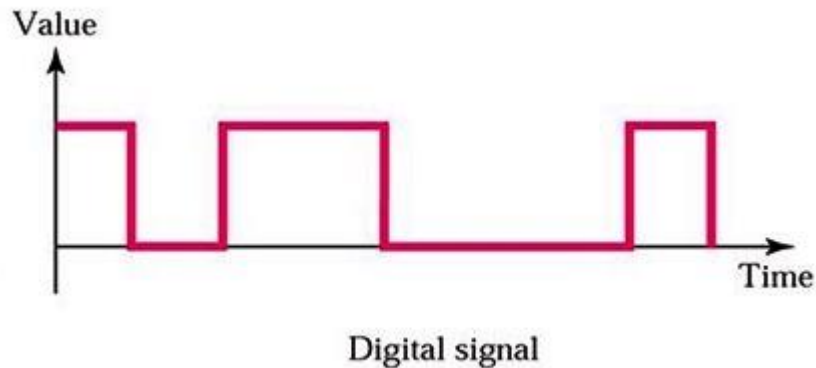
## 아날로그와 디지털



# I 기초

## 아날로그와 디지털

- 디지털 값: 0 또는 1
- 컴퓨터는 0 또는 1을 인식
- 컴퓨터는 전자기기이므로 전원이 공급됨
- 0 또는 1이라는 값은 어디서 오는 것인가

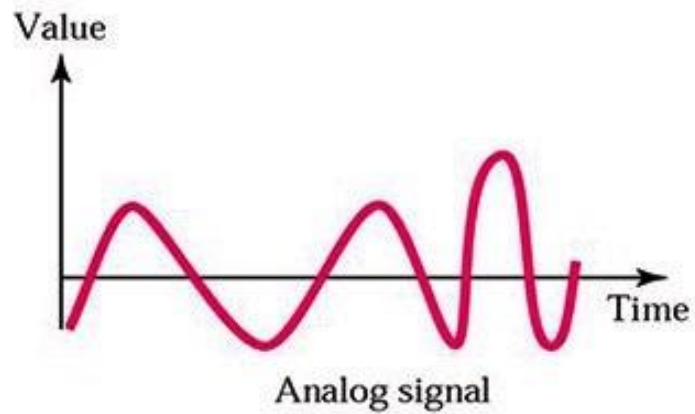
















0: 0  
1: 3.3v, 5v, ...

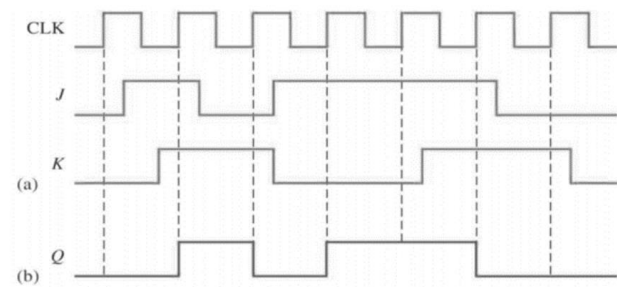
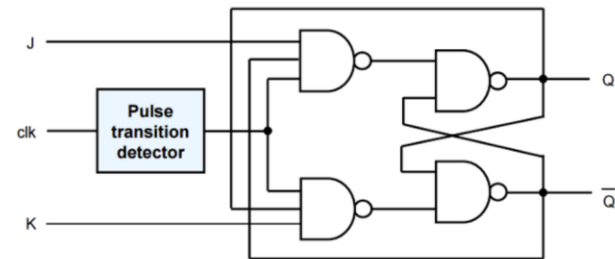
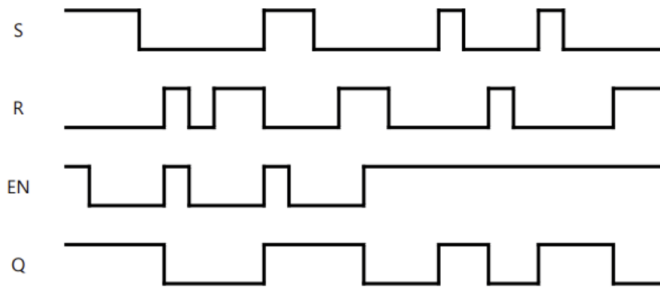
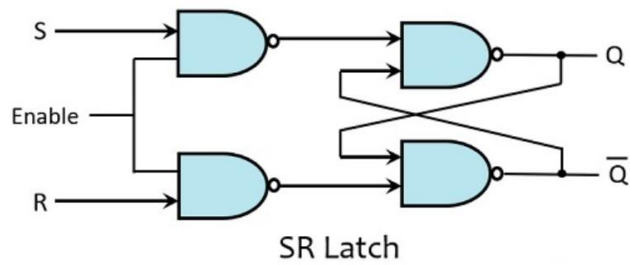


# 기초

## 아날로그와 디지털



Type 종류	Distinctive shape 독특한 모양 (IEEE Std 91/91a-1991)	Rectangular shape 네모 모양 (IEEE Std 91/91a-1991 IEC 60617-12 : 1997)	Boolean algebra between A & B A와 B의 부울대수	Truth table 진리표																		
NOT			$\bar{A}$ or $\sim A$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th></th><th>NOT A</th></tr><tr><td>0</td><td></td><td>1</td></tr><tr><td>1</td><td></td><td>0</td></tr></table>	INPUT		OUTPUT	A		NOT A	0		1	1		0						
INPUT		OUTPUT																				
A		NOT A																				
0		1																				
1		0																				
AND			$A \cdot B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	INPUT		OUTPUT	A	B	A AND B	0	0	0	0	1	0	1	0	0	1	1	1
INPUT		OUTPUT																				
A	B	A AND B																				
0	0	0																				
0	1	0																				
1	0	0																				
1	1	1																				
OR			$A + B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	INPUT		OUTPUT	A	B	A OR B	0	0	0	0	1	1	1	0	1	1	1	1
INPUT		OUTPUT																				
A	B	A OR B																				
0	0	0																				
0	1	1																				
1	0	1																				
1	1	1																				
NAND			$\overline{A \cdot B}$ or $A \uparrow B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A NAND B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A NAND B	0	0	1	0	1	1	1	0	1	1	1	0
INPUT		OUTPUT																				
A	B	A NAND B																				
0	0	1																				
0	1	1																				
1	0	1																				
1	1	0																				
NOR			$\overline{A + B}$ or $A \downarrow B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A NOR B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A NOR B	0	0	1	0	1	0	1	0	0	1	1	0
INPUT		OUTPUT																				
A	B	A NOR B																				
0	0	1																				
0	1	0																				
1	0	0																				
1	1	0																				
XOR			$A \oplus B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A XOR B	0	0	0	0	1	1	1	0	1	1	1	0
INPUT		OUTPUT																				
A	B	A XOR B																				
0	0	0																				
0	1	1																				
1	0	1																				
1	1	0																				
XNOR			$\overline{A \oplus B}$ or $A \odot B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A XNOR B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	INPUT		OUTPUT	A	B	A XNOR B	0	0	1	0	1	0	1	0	0	1	1	1
INPUT		OUTPUT																				
A	B	A XNOR B																				
0	0	1																				
0	1	0																				
1	0	0																				
1	1	1																				



# I 개요

## 2진수와 16진수

- 비트 표현
- 2진수: 0과 1로 구성
- 16진수: 0 ~ 9, A ~ F로 구성
- 예 16비트: 주소, 데이터, 명령, ...

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

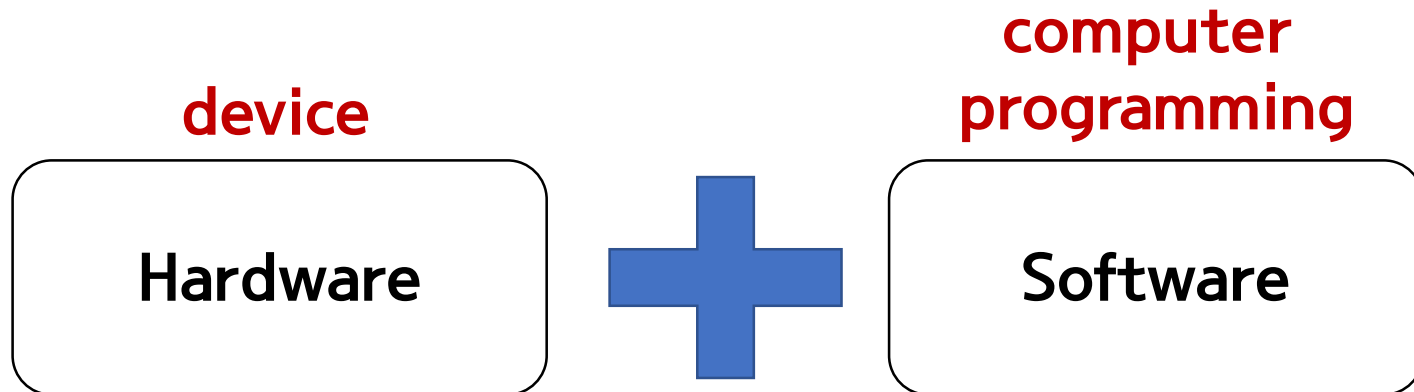
2진수: 1111 1111 1111 1111

10진수: 65,535

16진수: FFFF

# I What is Computer?

- A computer is a **device** that can be instructed to carry out sequences of arithmetic or logical operations automatically via **computer programming**. (Wikipedia.org)
- Computer의 구성



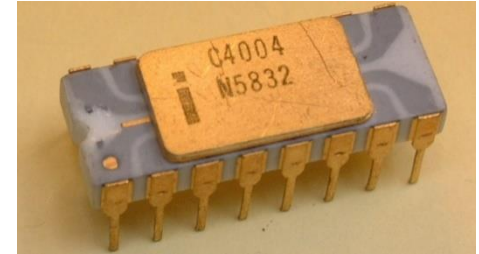


# What is Computer?

- Hardware

- CPU (Central Processing Unit)

- 프로그램의 연산을 실행하는 핵심적인 제어장치
    - 입력, 기억, 명령어 해석, 연산, 출력 기능 수행
    - 컴퓨터의 두뇌



- (Main) Memory

- 데이터 및 프로그램이 존재하는 공간
    - 수행중인 프로그램 코드가 존재



- Input/Output

- Input: 데이터 입력을 받는 hardware (키보드, 마우스)
    - Output: 데이터를 출력하는 hardware (모니터, 프린터)
    - Input/Output : 정보의 입출력이 가능한 hardware (HDD, 통신)

# | What is Computer?

- System Software

- Operating System: 컴퓨터 시스템을 구성하는 하드웨어와 사용자 또는 컴퓨터에서 실행되는 응용프로그램의 중간에 위치하여 사용자들이 보다 쉽고 간편하게 컴퓨터 시스템을 이용할 수 있도록 **컴퓨터 시스템을 제어하고 관리하는 시스템 소프트웨어**

eg) Windows 10, Linux, Android, Mac OS, DOS, UNIX, ...

- Operating System 역할

- 컴퓨터 하드웨어의 효율적 관리
- 사용자가 편리하게 컴퓨터 시스템을 사용할 수 있도록 사용자 Interface를 제공
- 사용자와 하드웨어 사이의 중간대화 통로 역할
- 메모리 관리와 입·출력 작업 보조

# I What is Computer?

- Programming Language
  - Machine Language
    - CPU가 직접 받아들일 수 있는 언어
    - CPU마다 언어가 다름
  - Assembly Language
    - Machine Language에 사용자가 사용하기 편하게 도와주는 software
    - Assembler를 통해 Machine Language로 변환
    - CPU마다 언어가 다름
  - High Level Language (C, C++, JAVA 등)
    - 사용자가 쉽게 프로그래밍 하도록 도와주는 software
    - CPU에 독립적
    - Compiler 혹은 Interpreter를 통해 Machine Language로 변환

# I What is Computer?

- Application Software
  - Programming Language를 이용하여 만들어짐
  - Operating System 위에서 수행되는 모든 Software

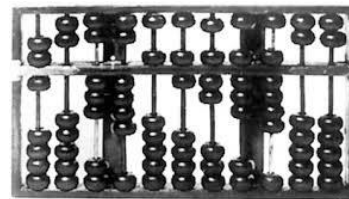
# 컴퓨터의 역사

- 기원전 3000년 수판

- 고대인들은 셈을 하기 위하여 손과 손가락을 사용
- 조약돌, 조개껍질, 염주알(beads)이 쓰이기도 했으며 후에 주판으로 발전
- 주판은 기원전 3천년 경에 메소포타미아(Mesopotamia)에서 쓰인 이래 세계 여러 나라에서 쓰이게 되었으며, 지금까지도 일부 국가에서 사용

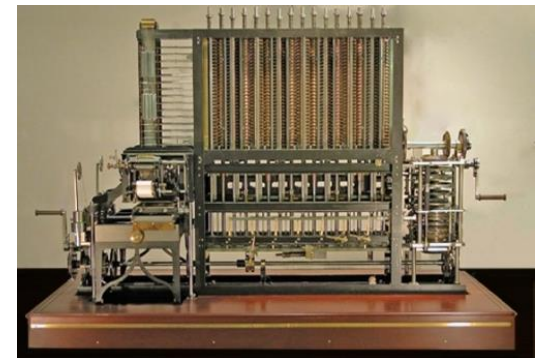
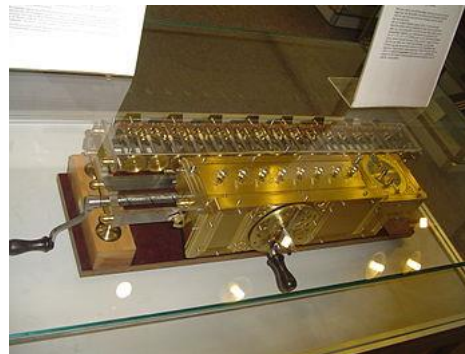
- 17세기 프랑스 Pascal 탁상용 계산기

- 기계식 덧셈기
- 이 계산기는 0 에서 부터 9까지 표시할 수 있는 10개의 톱니를 가진 톱니 바퀴가 여러 개 있어서 이들로써 가감산을 하도록 만들어 졌음



# 컴퓨터의 역사

- 라이프니츠의 계산기 (1673년)
  - 기계식 계산기
  - 덧셈, 뺄셈, 곱셈, 나눗셈이 가능
- 베비지의 계산기 (1822년)
  - 범용적인 자동 차축 계산기로 오늘날의 계산기와 비슷한 제어, 산술연산, 기억장치, 입출력장치를 가짐
  - 실제 제작되지 못함



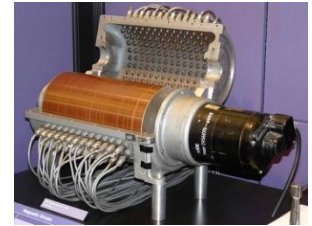
# 컴퓨터의 역사

- 미국 최초 전자 계산장치 ENIAC (1946년)
  - ENIAC: Electronic Numerical Integrator and Calculator
  - 18,800개의 진공관을 사용
  - 최초로 재프로그래밍이 가능한 전자식 계산기
  - 초당 5000번의 연산이 가능



# Computer의 세대별 분류

- 제1세대 computer(1951년~1958년)
  - 데이터의 저장과 처리에 진공관 사용
  - 주기억장치에 자기 드럼 사용
  - 입출력 보조기억 장치로 천공카드 사용
  - 프로그램은 기계어를 사용
  - 소프트웨어보다 하드웨어 개발에 중점
  - EDVC
    - 프로그램 내장 방식을 최초로 도입(폰 노이만)
  - 1세대 범용 컴퓨터
    - 모델명 701 (IBM) 상업용 컴퓨터
    - IBM650





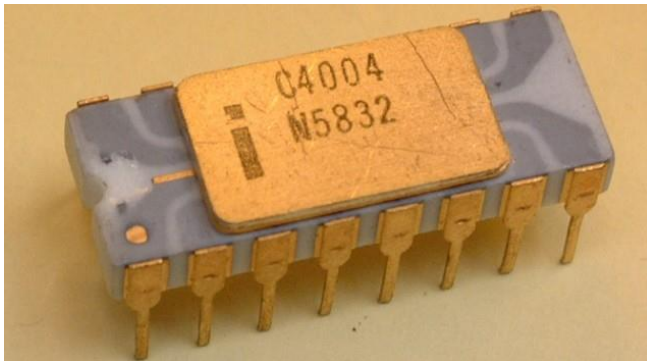
# Computer의 세대별 분류

- 제2세대 computer(1959년~1963년)
  - 회로소자로 트랜지스터 사용
  - 주기억장치로 자기 코어가 사용
  - 보조기억 장치로 자기 드럼, 자기 디스크가 사용
  - 입출력 장치로는 자기 테이프와 종이카드가 사용
  - 계산속도는 백만분의 1초 단위 정도 까지 향상



# Computer의 세대별 분류

- 제3세대 computer(1964년~1970년)
  - 중앙컴퓨터에 집적회로(IC)를 사용
  - 처리 장치는 소형화되고 기억 용량은 더욱 커짐
  - 다양한 소프트웨어 동작이 가능
  - 관리/처리 프로그램 및 사용자 프로그램 등 소프트웨어 체계가 확립
  - 다중 프로그래밍 시스템 및 시분할 시스템 개발



# Computer의 세대별 분류

- 제4세대 computer(1971년~)

- 기억소자로 (초)고밀도 집적회로(LSI, VLSI) 사용

- 연산속도의 증가

- 알테어 8800 (1970년 중반)

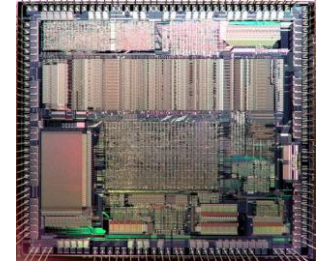
- 최초의 상업적인 개인용 컴퓨터

- 애플컴퓨터 (1977년)

- 스티브 잡스와 스테픈 워즈니악

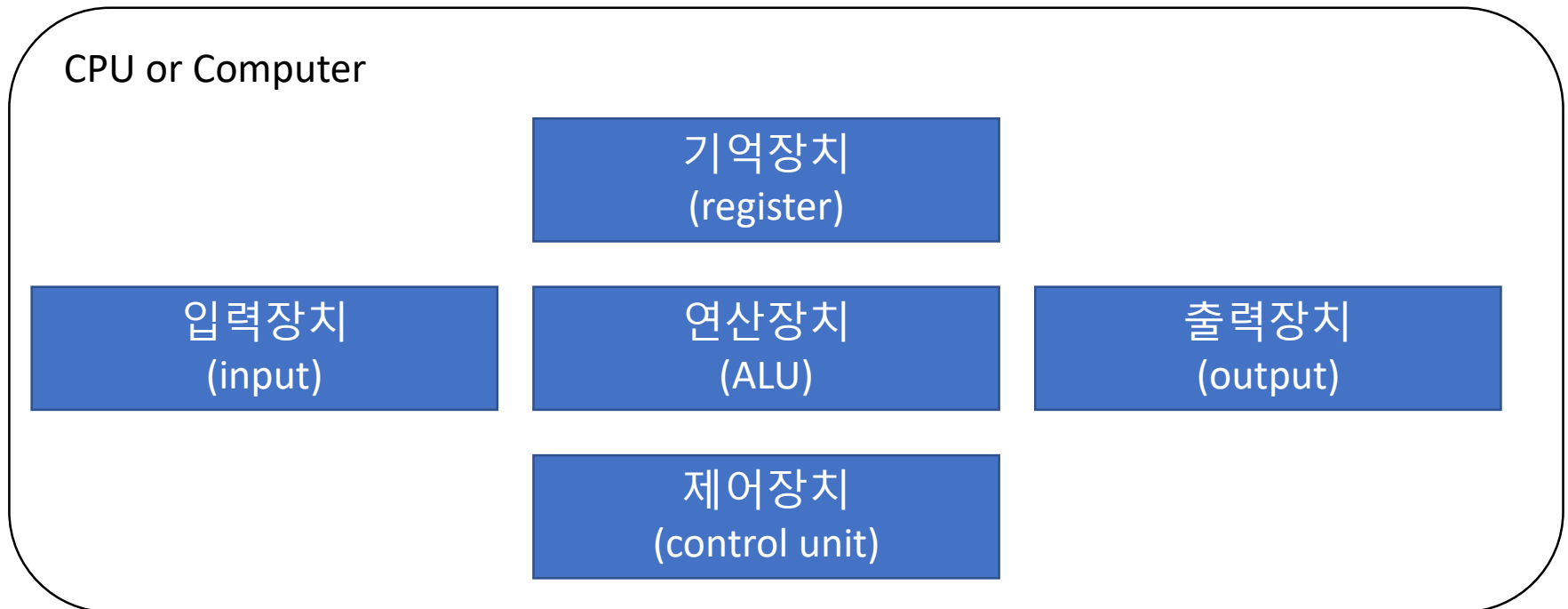
- IBM PC (1981년)

- 데스크 탑 컴퓨팅을 통해 개인용 컴퓨터의 표준으로 자리 매김



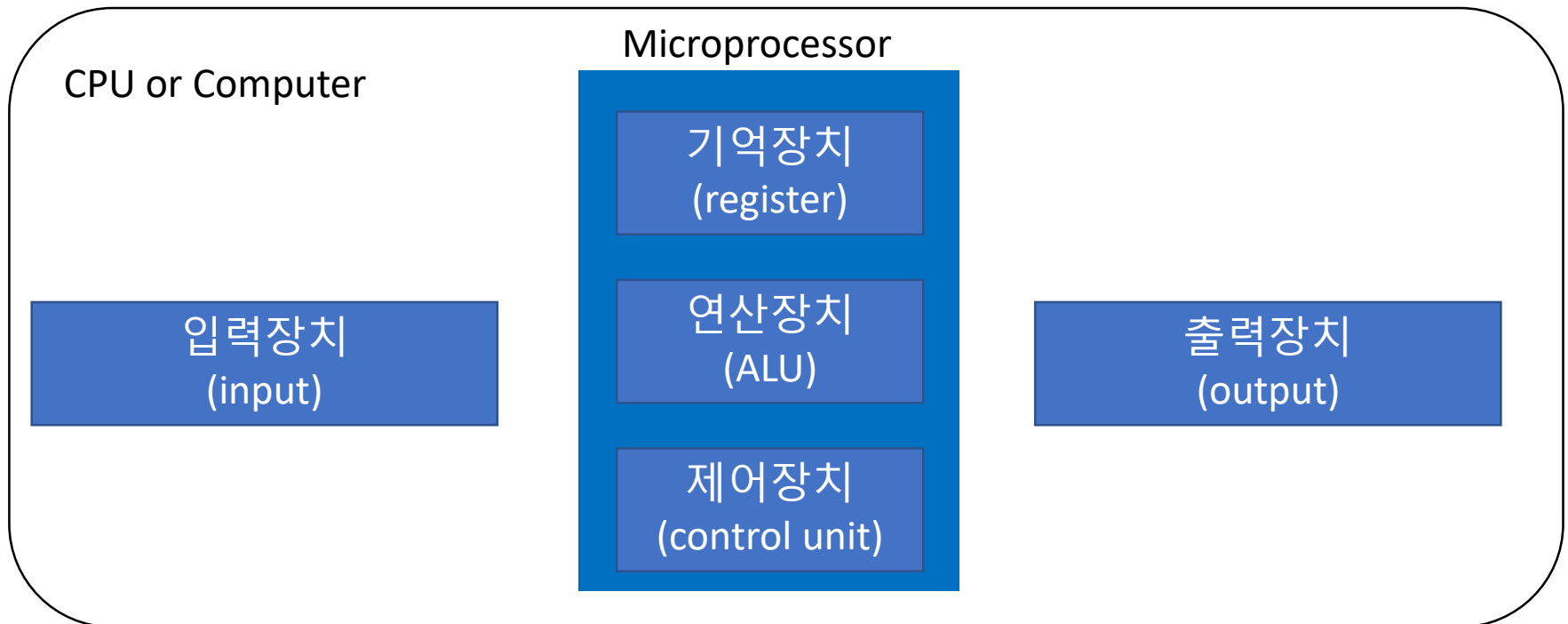
# Computer

- Computer(or CPU)의 기본 구성
  - 기억장치, 연산장치, 제어장치
  - 입력장치, 출력장치



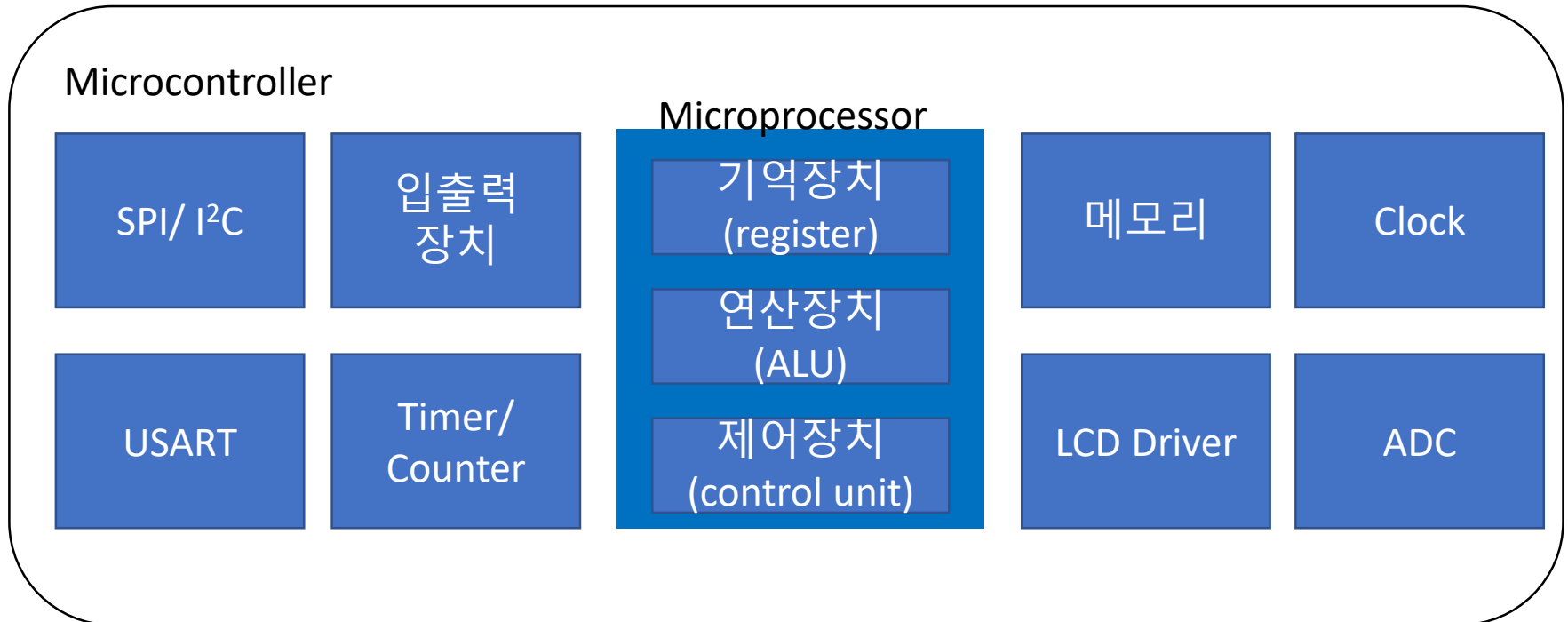
# Microprocessor

- Microprocessor
  - 기억장치, 연산장치, 제어장치를 1개의 chip으로 구성한 것
  - Microprocessor = CPU on Chip

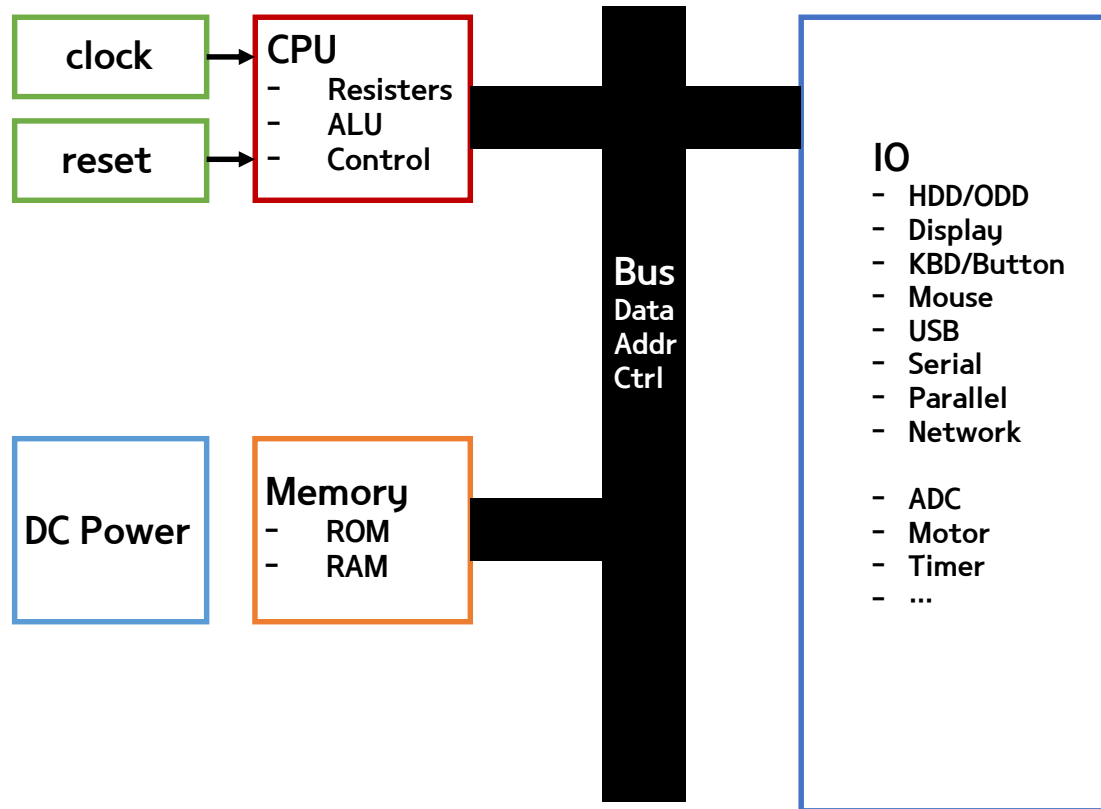


# Microprocessor

- Microcontroller
  - Microcontroller = (Microprocessor + I/O + memory + peripherals) on Chip



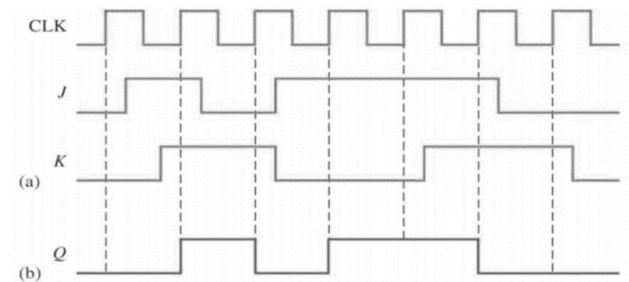
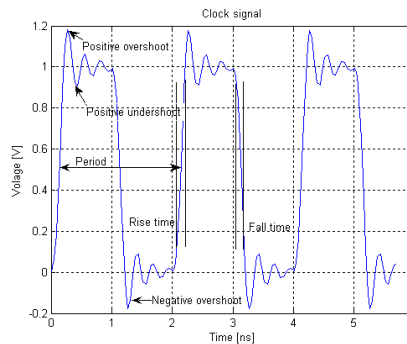
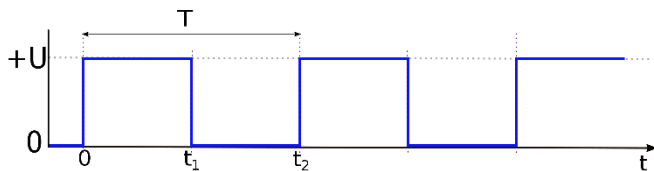
# Computer



# Computer

## Clock

- 클럭 신호는 논리상태 H(high, 논리 1)와 L(low, 논리 0)이 주기적으로 나타나는 방형파(square wave) 신호
- 1초동안 방형파를 몇 번 만들 수 있는지 → 속도
- 많은 경우 전자공학의 디지털 회로에서 클럭 신호에 맞추어 신호의 처리를 하는 동기 처리를 위해 사용





# I 관련 용어 및 기능

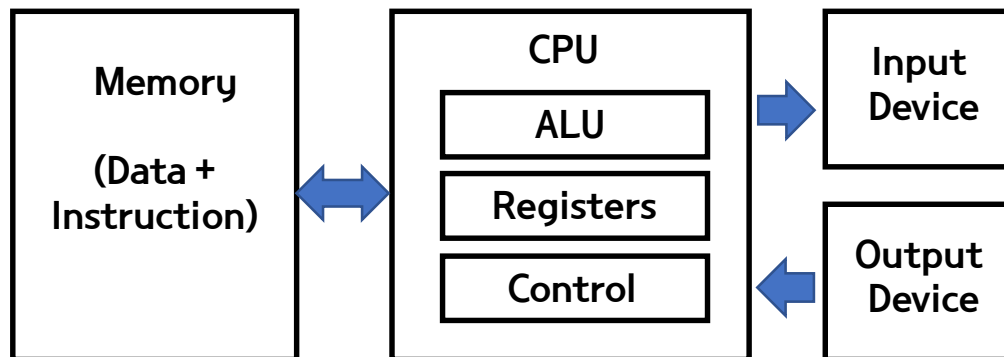
- CPU (Central processing unit): ALU + Registers + Control Unit
- 산술 논리 연산 장치(ALU : Arithmetic Logic Unit)
  - 가산이나 승산 등의 산술 연산과 AND 조작과 같은 논리 연산을 수행
- 레지스터(Register)
  - 프로그램의 실행 중에 데이터를 보관하는 작은 메모리로 고속 액세스 가능
- 제어 장치(Control Unit)
  - 명령어를 해석하고 그것을 실행하는데 필요한 컴퓨터 내부의 각 장치 사이의 데이터 흐름을 제어
- 버스(Bus)
  - 마이크로프로세서와 각 장치들이 서로 정보를 교환하기 위해 필요한 전송로
    - 주소 버스 : 메모리 내의 특정 장소나 입출력 장치의 특정 포트(port)를 지정하는 주소가 실린다.
    - 데이터 버스 : 각 장치 간에 주고받는 정보가 실린다.
    - 제어 버스 : CPU 내부 또는 외부로부터 시스템 동작을 제어하는 신호가 실린다.

# Computer

## CPU설계 방식에 따른 구분

- Von Neumann 구조

- 데이터 영역과 프로그램 영역의 물리적인 구분이 없고 버스 크기도 동일
- 데이터와 프로그램 코드를 동일하게 취급하므로 데이터도 프로그램 코드에 함께 들어감
- 마이크로프로세서 속도가 빠를수록 고성능, 대표적인 예. 일반 데스크톱 PC

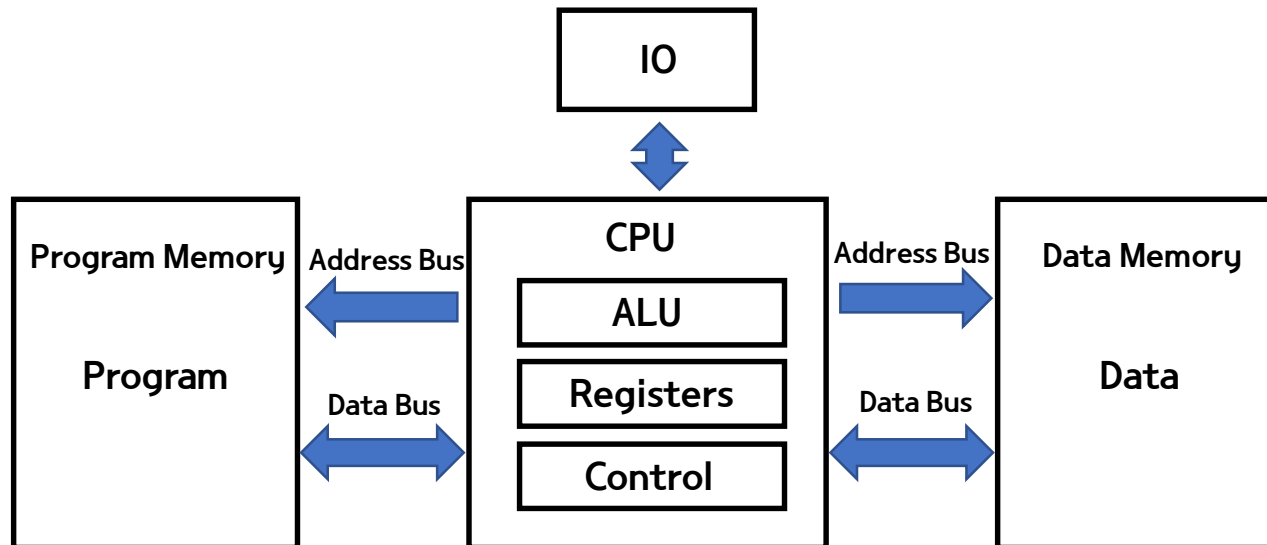


# Computer

## CPU설계 방식에 따른 구분

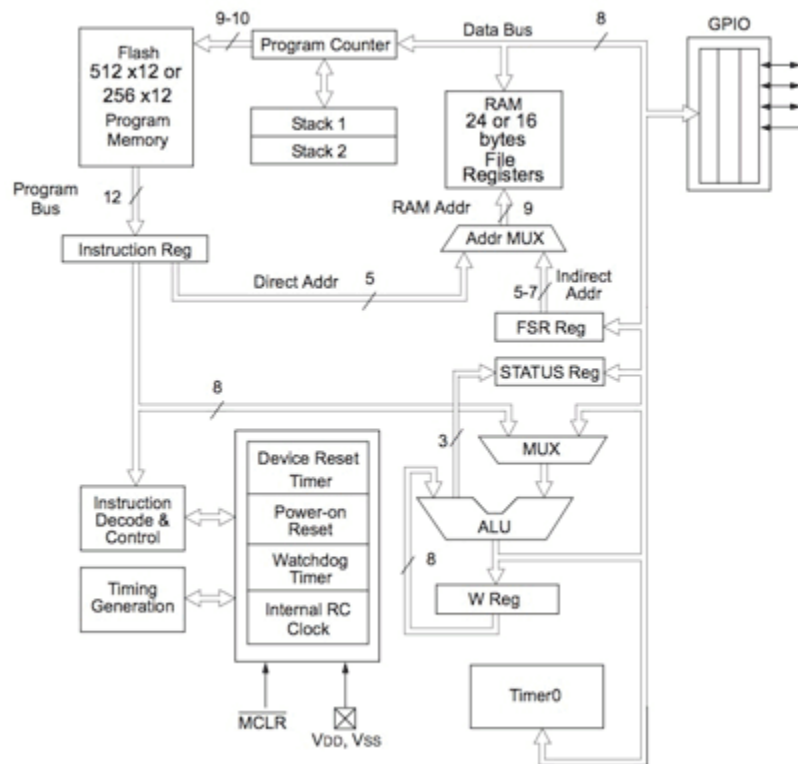
- Harvard 구조

- 데이터 영역과 프로그램 영역이 물리적으로 구분되어 있고 각 버스의 크기도 다를 수 있음
- 데이터와 명령어를 분리해서 처리하며, 대표적인 예는 8051, PIC, AVR

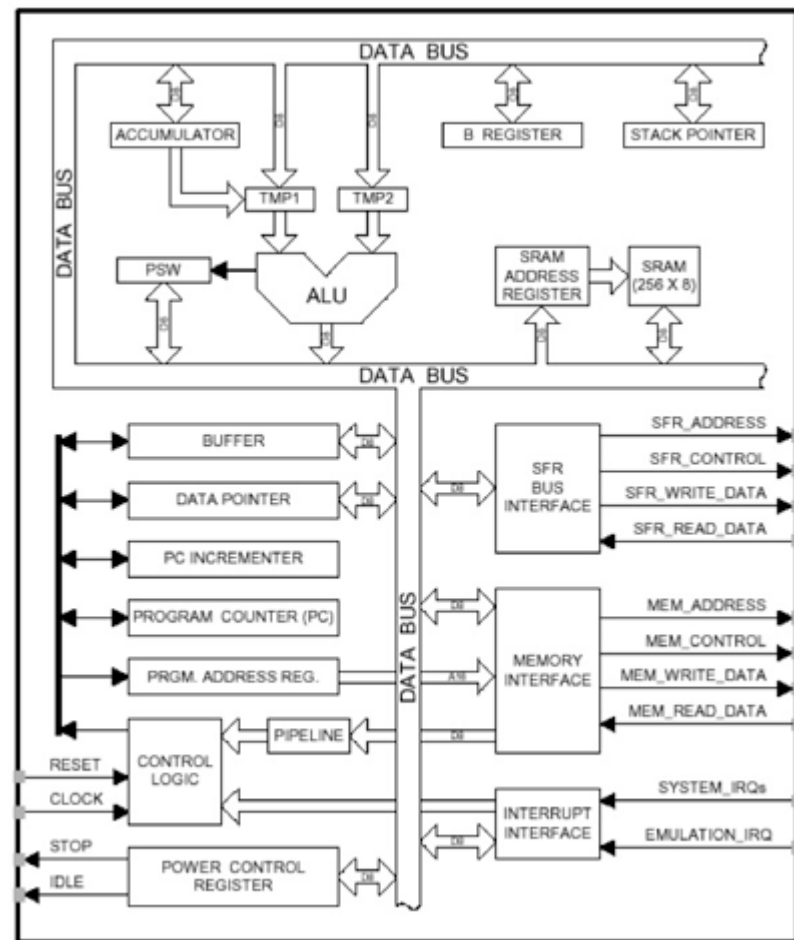


# Computer

## CPU설계 방식에 따른 구분



Harvard

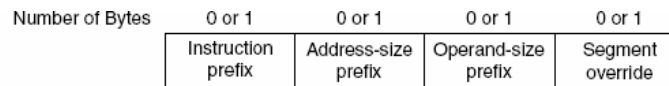


Von Neumann

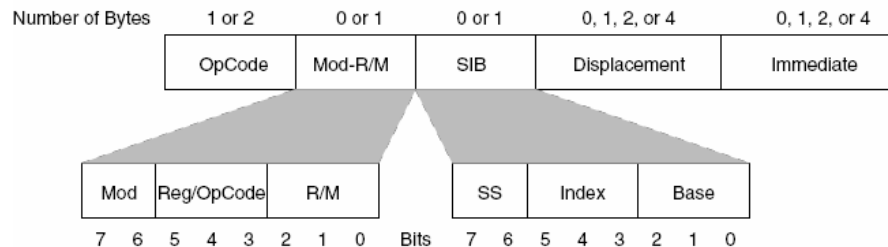
# Computer

## 명령어 구성 방식에 따른 구분

- CISC (Complex Instruction Set Computer) 구조
  - 가변길이 명령어를 사용한다 (1byte~8byte)
  - 프로그램 내장방식으로 설계 되어있다
  - 명령어가 복잡하고 개수가 많다
  - 명령어 해석기가 마이크로 프로그램으로 구성되어 있다



(a) Optional instruction prefixes

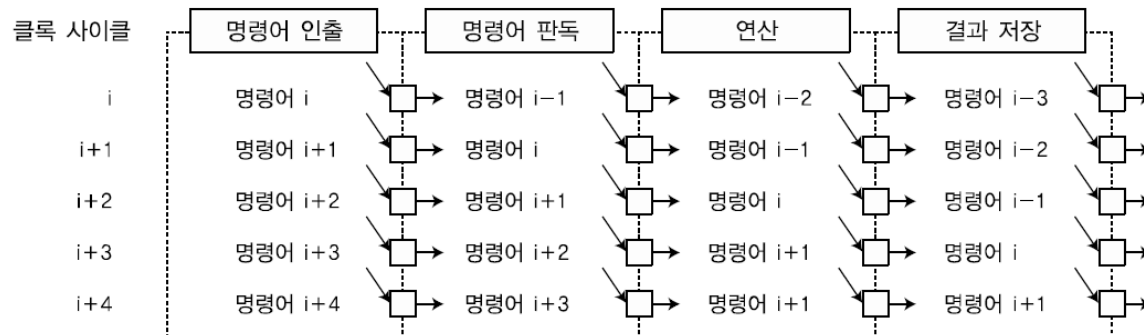


(b) General instruction format

# Computer

## 명령어 구성 방식에 따른 구분

- RISC (Reduced Instruction Set Computer)
  - 명령어가 고정된 고정길이 명령어 사용
  - 일반적으로 하버드 구조로 구성되어 있다
  - 명령어 개수가 적다
  - 속도가 빠르다. 명령어의 단일 사이클 실행
  - 명령어 해석기가 하드와이어(hardwire)로 구성되어 있다
  - 파이프라인 기법(Pipelining)을 이용하여, 한 사이클에 한 명령어 실행



# Computer

## 명령어 구성 방식에 따른 구분

- RISC (Reduced Instruction Set Computer)

- 고정된 짧은 길이의 명령어

- CPU 시스템 버스 크기와 같게 명령어의 크기를 고정해서 단일 클록에 명령어를 명령어 레지스터로 인출

- 단순한 어드레싱 모드 사용

- 복잡한 어드레싱 모드의 사용은 일관적인 명령어 길이를 보장할 수 없음
    - 단순한 어드레싱 모드의 사용으로 오퍼랜드 인출 절차를 통일
    - load와 store 메모리 참조 명령에 단순한 어드레싱 모드를 사용하면, CPU 내부 하드웨어 복잡도를 줄이고, 명령어의 길이를 동일하게 유지할 수 있음

# | Computer

## 명령어 구성 방식에 따른 구분

- RISC (Reduced Instruction Set Computer)
  - 간단한 연산 동작의 적은 명령어 셋
    - 명령어에서 간단한 산술과 논리 연산 동작을 사용하여, 연산 단계에서 짧은 시간에 수행
    - 구현되는 명령어 개수가 적으면, 각 명령어의 기능을 만족하기 위한 하드웨어 복잡도를 줄일 수 있음
  - 현대에는 마이크로프로세서가 고성능화 되면서, RISC와 CISC의 구별이 점점 모호해지고 있음



# Microprocessor

## Microcontroller의 분야별 응용

산업	<ul style="list-style-type: none"><li>• 모터 제어</li><li>• 로봇 제어</li><li>• 프로세스 제어</li><li>• 수치제어</li><li>• 지능형 변환기(transducer)</li></ul>	계측	<ul style="list-style-type: none"><li>• 의료용 계측기</li><li>• 가스 크로마토 그래프</li><li>• 오실로스코프</li></ul>
가전제품	<ul style="list-style-type: none"><li>• 비디오 레코더</li><li>• 레이저 디스크</li><li>• 비디오 게임기</li></ul>	유도제어	<ul style="list-style-type: none"><li>• 미사일 제어</li><li>• Torpedo 유도 제어</li><li>• 지능형 무기</li><li>• 우주선 유도 제어</li></ul>
데이터 처리	<ul style="list-style-type: none"><li>• 플로터(plotter)</li><li>• 복사기</li><li>• 하드디스크 구동장치</li><li>• 테이프 구동장치</li><li>• 프린터</li></ul>	자동차	<ul style="list-style-type: none"><li>• 점화 타이밍 제어</li><li>• 연료 분사 제어</li><li>• 변속기 제어</li><li>• ABS 제어</li><li>• 열 방사 제어</li></ul>
통신	<ul style="list-style-type: none"><li>• 모뎀</li><li>• 지능형 line card 제어</li></ul>		

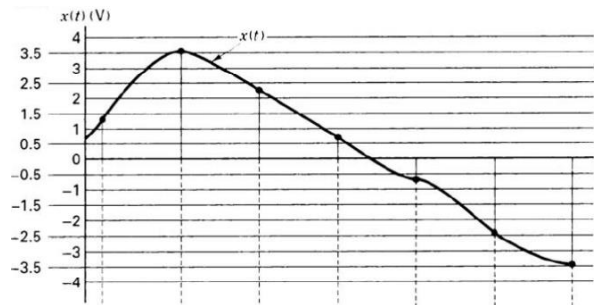
# Microprocessor

## Microcontroller의 성능에 따른 응용

4비트 CPU	<ul style="list-style-type: none"><li>• 전자렌지, 전기밥솥, 가스 오븐, 전기세탁기 등 가전제품</li><li>• 휴대용 음향기기</li><li>• 자동차용 라디오</li><li>• 게임기</li></ul>
8비트 CPU	<ul style="list-style-type: none"><li>• 단말기</li><li>• 계측기기</li><li>• 고급 탁상계산기</li><li>• 학습기</li><li>• 재고 관리기</li><li>• 감시장치</li></ul>
12비트 CPU	<ul style="list-style-type: none"><li>• 자동차 전자 장치</li><li>• 계측</li><li>• 상수도, 가스 등의 telemeter</li></ul>
16비트 이상의 CPU	<ul style="list-style-type: none"><li>• 프로세서 제어</li><li>• 기타 복잡한 기계나 시스템 제어</li></ul>

# Microprocessor

- 예. 어떤 데이터를 측정했을 때  $-4 \sim 4$  범위에서 측정되었음: 아날로그 신호(데이터)
- 컴퓨터는 3.1, 3.111, 3.111111, ... 등을 모두 표현할 수 없을 것: 개수가 무한대
- 만약,  $-4 \sim 4$ 를 9등분하면,  $-4, -3, -2, \dots, 2, 3, 4$ 로 9개 숫자가 있을 것
- $-3.6$ 는  $-4$ 로,  $-3.2$ 는  $-3$ 으로 표현할 수 밖에 없음
- 만약,  $-4 \sim 4$ 를 90등분하면,  $-4, -3.9, \dots, 3.9, 4$ 로 90개 숫자로 표현 가능
- 데이터에서 비트는 데이터를 얼마나 촘촘히 나누어 표현할 것인지를 결정

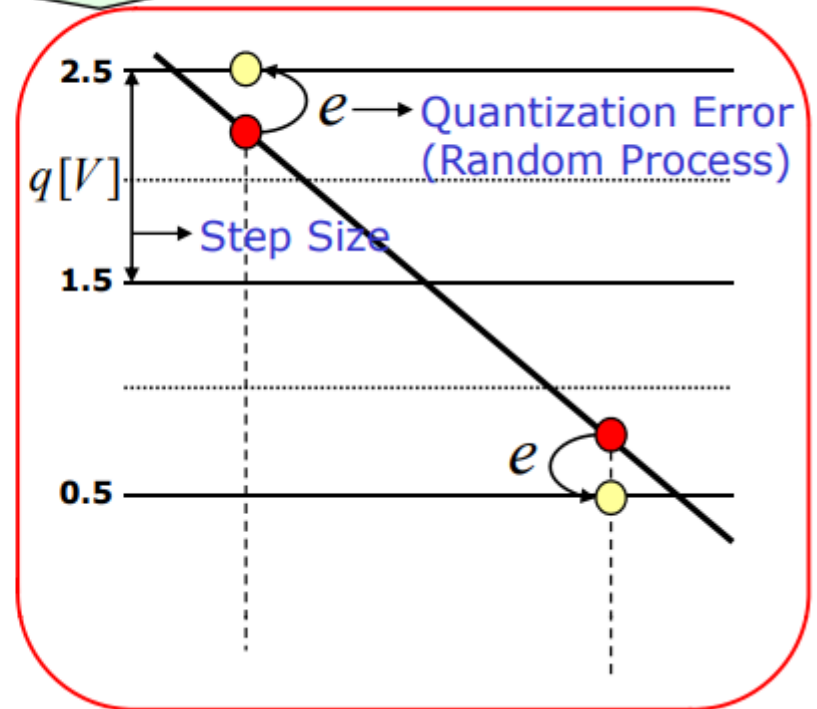
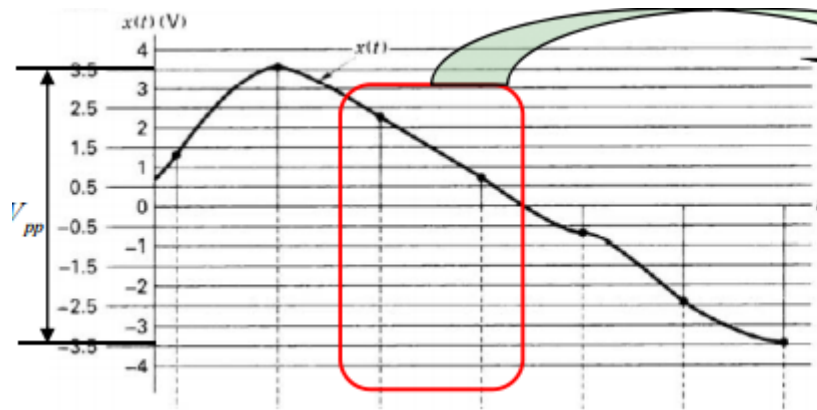


# Microprocessor

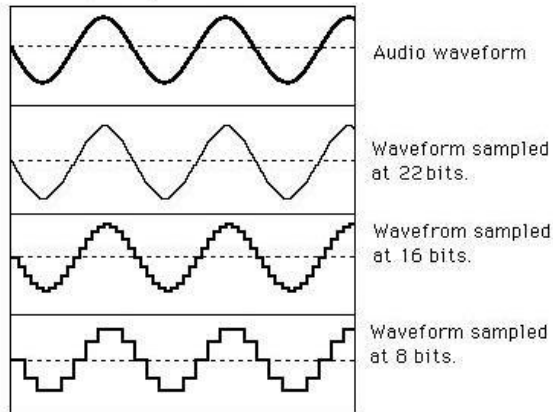
- 예를들어, char형 256개의 숫자표현 ( $2^8=256$ , 8비트, 1바이트)
- $-4v \sim 4v$  사이를 256등분하여 표현할 수 있을 것

-4	0	0	0	0	0	0	0	00
-3.9686	0	0	0	0	0	0	1	01
-3.9373	0	0	0	0	0	1	0	02
-3.9059	0	0	0	0	0	1	1	03
...	...							
4	1	1	1	1	1	1	1	FF

# Microprocessor



Sound quality and bits.



# Microprocessor

- 계열별 마이크로컨트롤러
  - 8비트 : 인텔의 8051 계열, 마이크로칩(Microchip)의 PIC 계열, 아트멜(Atmel)의 AVR 계열, 모토롤라의 68k 계열, 자이로그(Zilog)의 Z88 계열
  - 32비트 : 인텔 80960, 모토롤라 68332, ARM 계열, 프리스케일 ColdFire, MIPS
- 인텔의 8051 계열
  - 마이크로컨트롤러 초창기에 8비트 시장에서 큰 인기를 누린 인텔이 8051을 이용한 마이크로컨트롤러를 직접 생산했지만, 현재는 아트멜, NEC, 필립스(현 NXP) 등의 주요 업체가 호환 프로세서를 생산
  - 하지만 8비트 마이크로컨트롤러의 대표격인 8051 코어를 이용한 제품은 전자제품 등에서 아직도 많이 사용

# Microprocessor

- 마이크로칩(Microchip)의 PIC 계열
  - RISC 방식의 8비트 마이크로컨트롤러
  - 속도, 내부 메모리 용량, 내장 디바이스 특성에 따라 PIC12, PIC14, PIC16, PIC17, PIC18 계열이 있음
  - 동작 전원 범위(2.0V~ 6.0V)가 넓고, 소비 전류는 수 mA 이하이며, 파워 on Reset과 원 칩 타이머, ROM 등을 내장하고 있어 소형 제품을 만드는 데 적합
- 아트멜(Atmel)의 AVR 계열
  - ATiny, AT90S, ATmega 계열이 있음
  - 동작 전원 범위(1.8V~5.5V)가 넓고, 프로그램 코드를 저장할 플래시 메모리와 데이터를 저장할 EEPROM, SRAM 등 다양한 내부 메모리를 제공
  - 모든 계열이 ROM writer와 같은 별도의 장비 없이 PC에서 AVR의 내부 플래시 메모리로 프로그램을 저장하는 ISP(In-System Programming) 기능을 갖추고 있음

# Microprocessor

- 동작 속도 비교

- PIC : 5Mips(20MHz를 내부적으로 1/4 분주),
- 8051 : 2.5~3Mips(20M → 내부적으로 1/7~8 분주)
- AVR이 8Mips(8MHz를 내부 분주 않음)
- AVR이 빠르다고 볼 수 있음

- 명령어 개수 비교

- PIC : 약 35개,
- 8051 : 약 111개,
- AVR : 약 120개
- PIC는 명령어가 적어서 배우기 편하지만 프로그램 작성 시 어려움이 있음



# Microprocessor

- 프로그래밍 방식 비교

- PIC, 8051 : 일부 모델은 ROM 라이터와 자외선 에라이저(UV erasure)가 필요하지만 플래시 타입은 ISP로 편리하게 프로그래밍할 수 있음
- AVR : 모든 계열을 ISP 방식으로 프로그래밍할 수 있다는 장점이 있음

- 개발 언어 비교

- PIC : 어셈블리어가 좋고, C 언어는 하드웨어 구조상 문제점이 많고 생성하는 코드의 효율도 좋지 않음
- 8051 : 어셈블리어와 C 언어 모두 좋지만 C 언어는 느리다.
- AVR : 어셈블리어와 C 언어 모두 좋으며, C 언어의 경우 속도 차가 있지만 무시해도 좋을 정도다.
- 8051이나 PIC는 C 컴파일러가 고가지만 일부 기능을 제한하여 무료로 제공하는 버전이 있으며, AVR은 상용에 뒤지지 않는 무료 버전인 AVR-GCC가 있음
- 최근에 8051은 무료로 제공하는 SDCC도 있음

- ARM 코어

- 임베디드 시스템에서 저전력, 저발열, 고성능의 이점이 있어 많은 마이크로컨트롤러 제조업체로부터 주목을 받기 충분했음
- 이후 여러 마이크로컨트롤러 제조업체가 ARM 코어를 선택함에 따라, ARM 7을 필두로 ARM 9, ARM 11 등의 ARM 코어는 관련 업계의 폭넓은 지지를 받았음
- 현재 ARM 코어는 PDA나 PMP, 게임기 등의 저전력을 요구하는 소형기기에 사용되며 네트워크 장비인 IP 공유기나 라우터 등에서도 사용된다

# | CPU

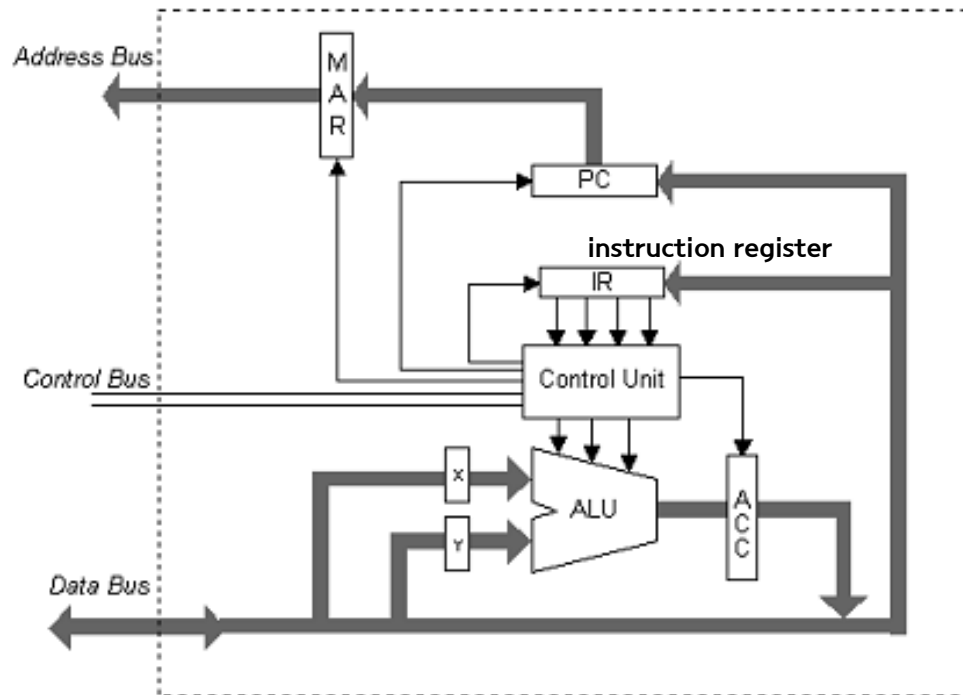
## Central Processing Unit

- Functional Unit + Register + Control Unit
- Functional Unit
  - ALU: Arithmetic and Logic Unit
  - 다양한 목적의 functional unit 포함할 수 있음 (예. 곱셈기 등)
- Register
  - Small, temporary storage
  - Operands and results of functional unit

## **Central Processing Unit**

- **Functional Unit + Register + Control Unit**
- **Control Unit**
  - **Orchestrates execution of the program**
  - **Program Counter (PC): contains the address of the next instruction to execute**
  - **Reads an instruction from memory (at PC)**
  - **Interprets the instruction**
  - **Generates signals that tell the other components what to do**
  - **Instruction may take many machine cycles to complete**

# CPU

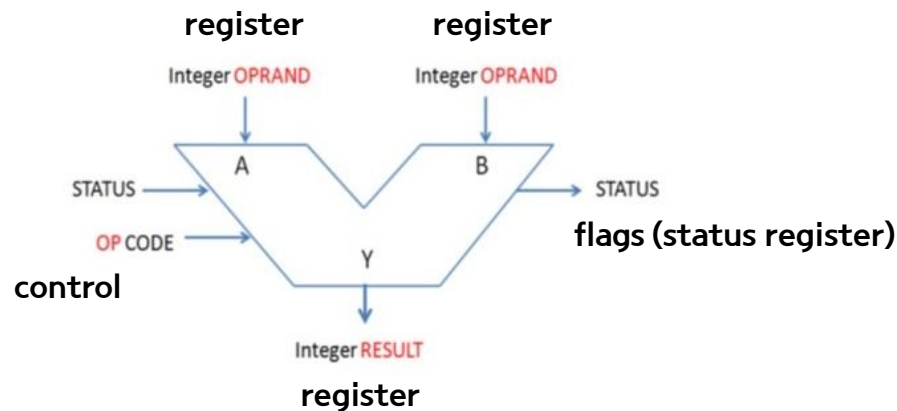


Simple CPU

# CPU

## ALU: Arithmetic and Logic Unit

- Does the calculation
- Integer arithmetic operations (addition, subtraction and sometimes multiplication and division, though this is more expensive)
- Bitwise logic operations (NOT, AND, OR, XOR)
- Bit-shifting operations



# I Registers

- CPU must have some working space (temporary storage)
- Number and function vary between processor designs
- General purpose register
  - CPU 연산을 빠르게 처리하기 위해 ALU와 직접 연결
  - 연산 대상이 되는 오퍼랜드 값을 가짐
- Special Purpose Register
  - 명령어를 실행할 때 필요한 범용 데이터가 아닌 특수한 데이터를 처리하기 위한 레지스터
  - 프로그램 카운터, 명령어 레지스터, 상태 레지스터, 메모리 주소 레지스터, 메모리 버퍼 레지스터 등

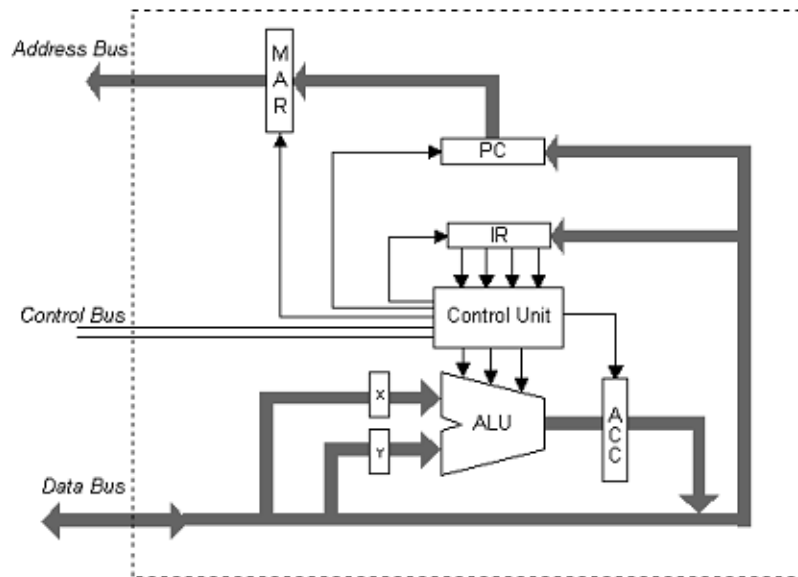
# | Registers

- 프로그램 카운터(PC, Program Counter)
  - 인출할 명령어가 있는 메모리의 주소를 갖는 특수 레지스터
  - 프로그램 메모리에서 한 개의 명령어 인출이 끝나면, 그 명령어의 크기가 더해진 값으로 자동 변경되어 다음 명령어 인출을 위한 주소를 가짐
- 명령어 레지스터(IR, Instruction Counter)
  - 프로그램 메모리에서 인출된 명령어를 기억하고 있는 특수 레지스터
  - 인출된 명령어에는 특정 비트에 연산 동작(opcode), 연산 대상(operand), 연산 결과(result) 어드레싱 모드 등의 정보를 설정

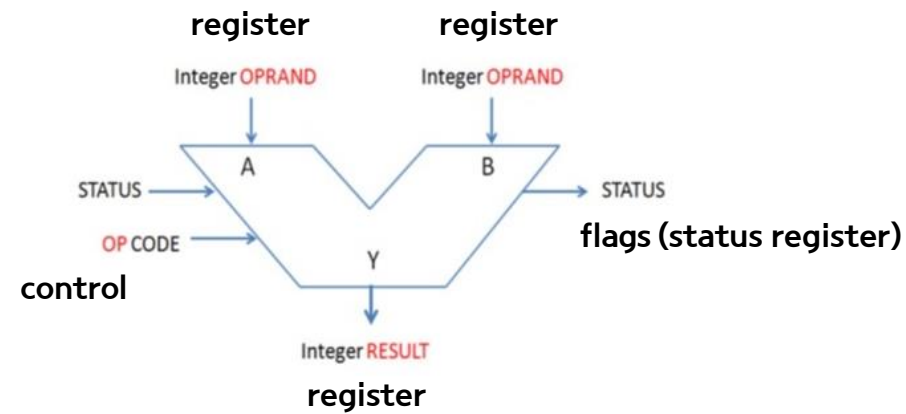


# | Registers

- 상태 레지스터(Status Register)
  - 명령어를 실행한 후의 연산 결과 정보를 기록
  - 기록된 상태 레지스터의 각 비트는 연속되는 다음 명령어 실행에 영향을 미침
  - carry, zero, negative, overflow, sign, interrupt, ...
- 메모리 주소 레지스터(MAR)와 메모리 버퍼 레지스터(MBR)
  - 데이터 인출/기록 과정에서 데이터 또는 주소를 임시로 저장하기 위한 특수 레지스터



Simple CPU



# I Instruction

- Fundamental unit of work
- Constituents
  - Opcode: operation to be performed
  - Operands: data/locations to be used for operation
  - eg) ADD R0, 3
- Encoded as a sequence of bits
  - Sometimes have a fixed length
  - Control unit interprets instruction  
Generates control signals to carry out operation
- Instruction Set Architecture (ISA)
  - Computer's instructions, their formats, their behavior

# I Instruction Set

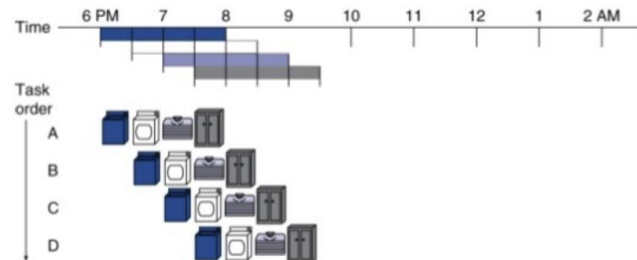
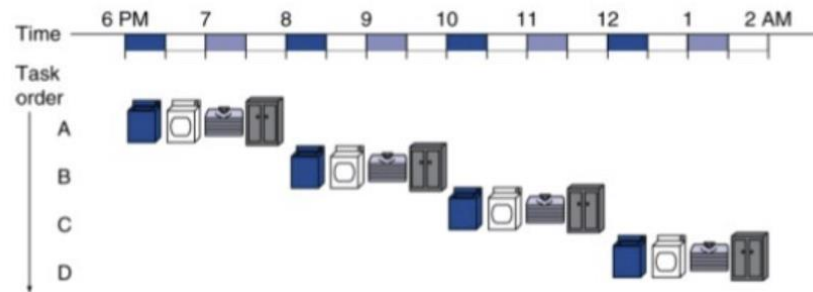
- Arithmetic instructions such as add and subtract
- Logic instructions such as AND, OR, XOR, and NOT
- Data instructions such as move, input, output, load and store
- Control flow instructions such as goto, if, call, and return

## 기본동작원리

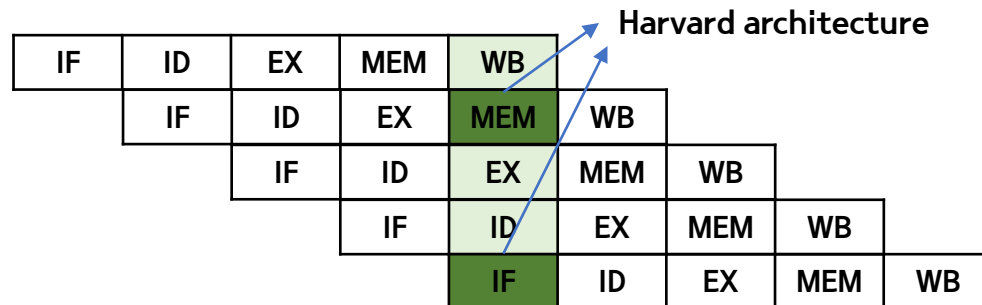
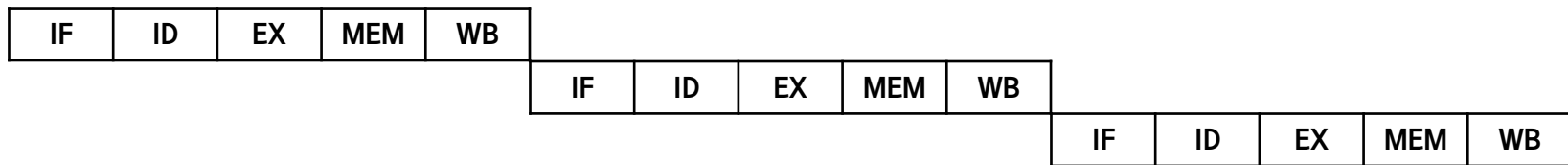
- Instruction Fetch (IF)
  - 프로그램 메모리에 있는 명령어를 인출(Fetch)하여, 명령어 레지스터(IR)에 넣는 단계
- Instruction Decode (ID)
  - 명령어 레지스터(IR) 정보를 해독하고 동시에 레지스터를 읽음
- Excute (EX)
  - 연산을 수행하거나 주소를 계산
- Memory access (MEM)
  - 데이터 메모리에 있는 피연산자에게 접근
- Writeback (WB)
  - 결과값을 레지스터에 씬

- RISC 구조로 실행 단계를 단축할 수 있음
  - 오퍼랜드 인출 단계를, 별도의 명령어 load
  - 결과 기록 단계를, 별도의 명령어 store
  - 단순해지는 인출-실행 사이클

- 세탁기 예.
- 세탁, 탈수, 옷개기, 옷장 넣기를 세탁의 한 사이클이라고 가정



다른 명령어가 빨리 끝나도 클럭 시간이 줄어들지 않는다

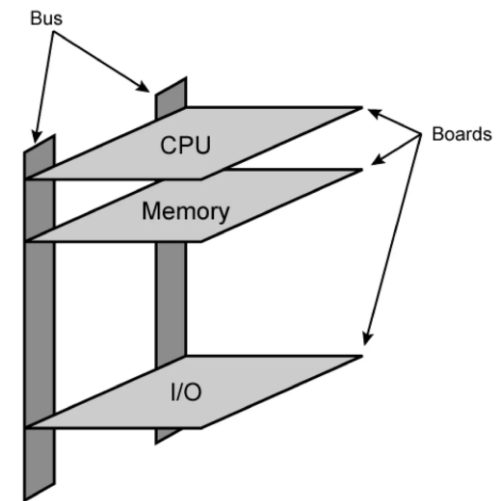
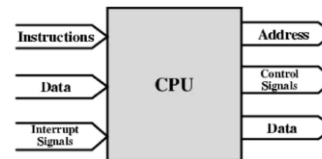
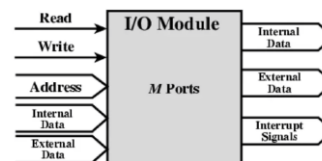
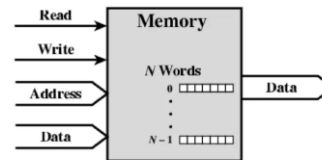
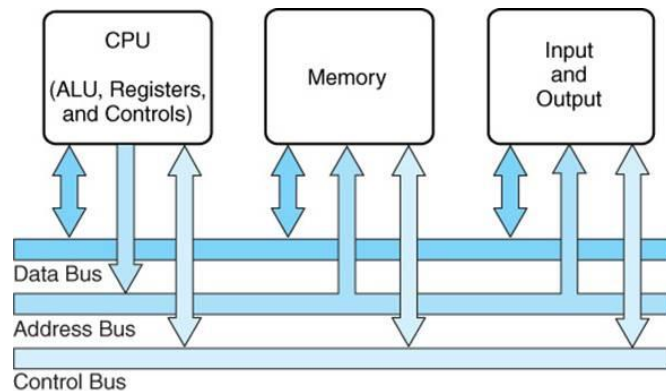


Pipelining!



# Bus

- All the units must be connected
- Different type of connection for different type of unit
  - CPU
  - Memory
  - IO



# | Bus

- Data
  - Carries data (including instructions)
  - Width is a key determinant of performance (8, 16, 32, 64 bits)
- Address
  - Identify the source or destination of data
  - CPU needs to read an instruction (data) from a given location in memory
  - Width determines maximum memory capacity of system
- Control
  - Memory read/write signal
  - Interrupt request
  - Clock signals

# Bus

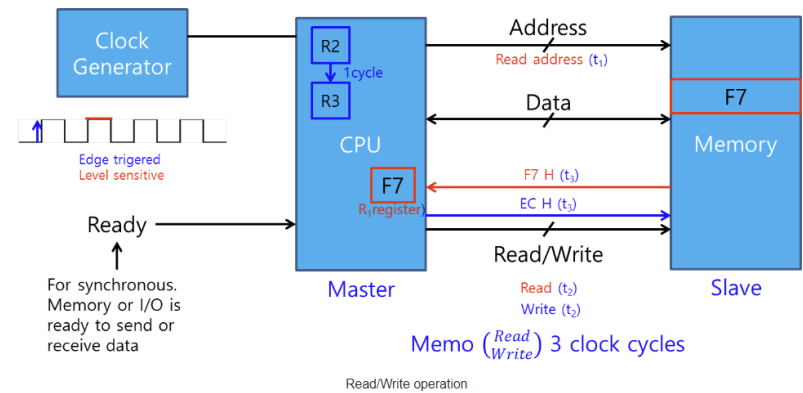
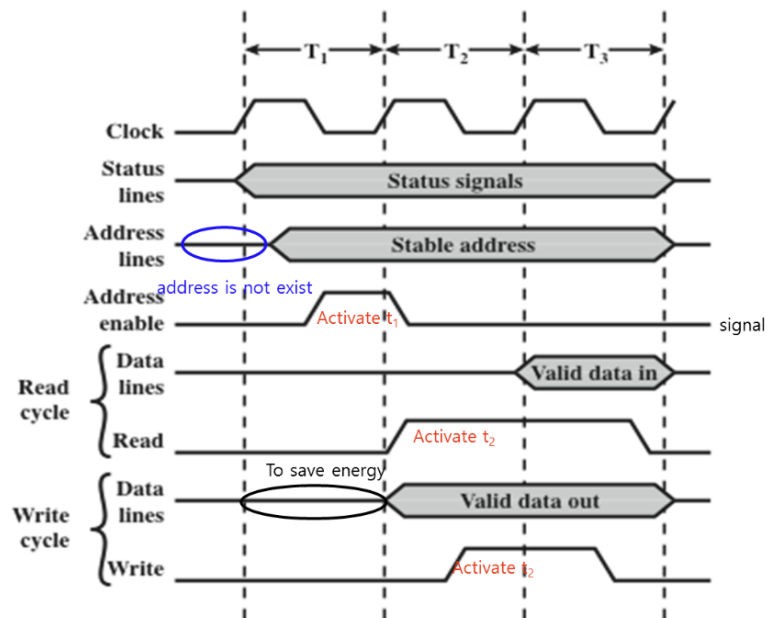


Figure 3.18 Timing of Synchronous Bus Operations

# | Memory

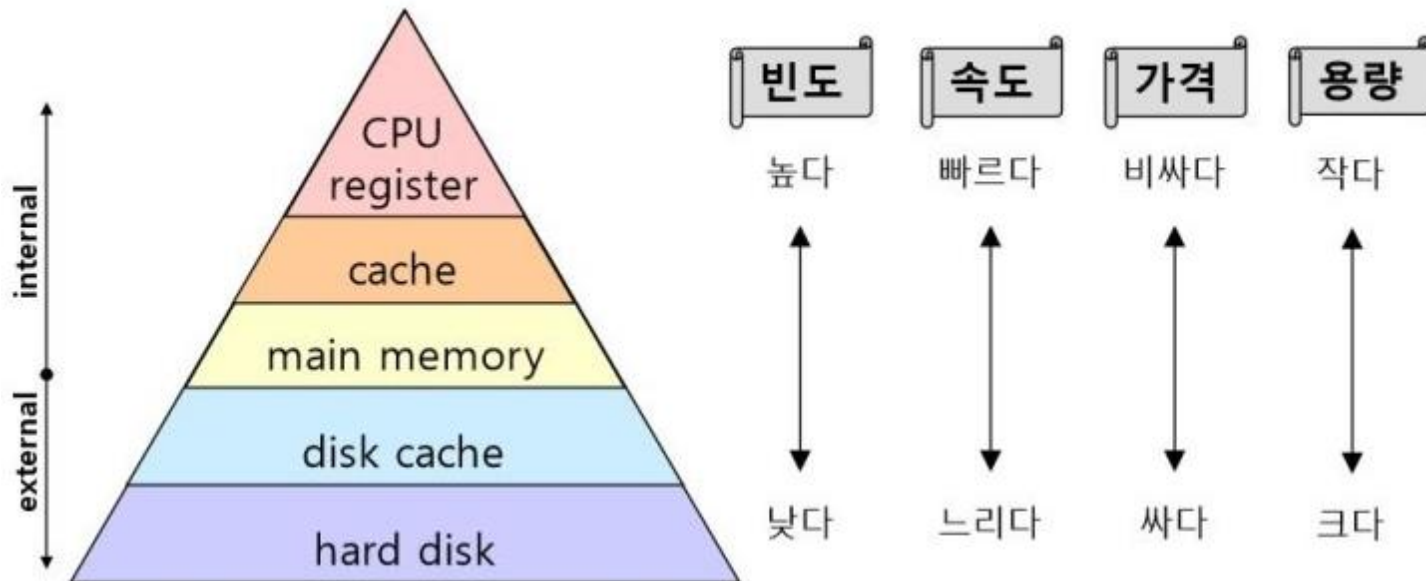
---

- Internal and external
- ROM (PROM, EPROM, FLASH) and RAM
- DRAM (SDRAM, DDR RAM) and SRAM
- Cache, Main memory virtual memory

# Memory

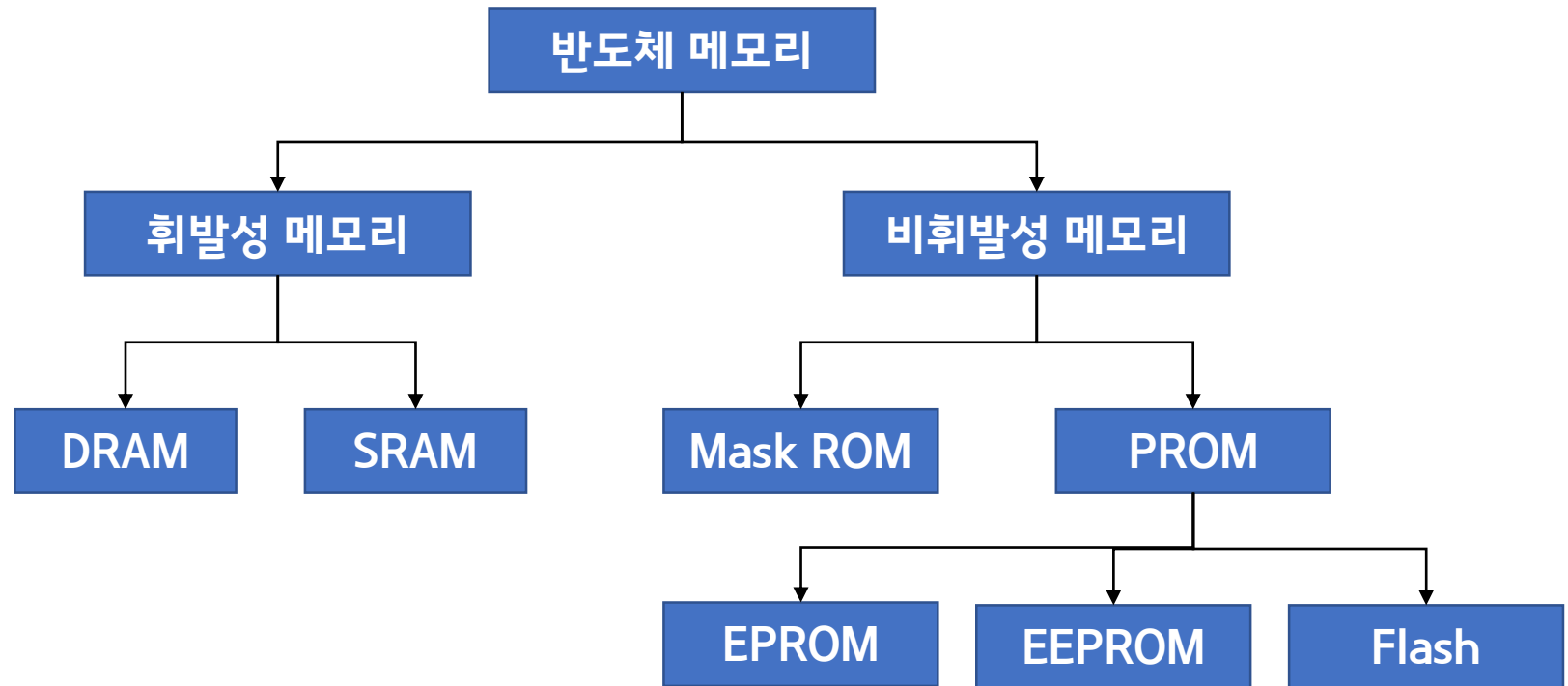
## Memory의 계층구조

- 사용빈도, 속도, 가격, 용량



# Memory

## Memory의 분류



# | Memory

## Memory의 분류

- 휘발성 메모리(volatile memory)
  - 일정한 시간이 지나거나 전원이 꺼지면 기록된 내용이 지워지는 메모리
  - RAM은 모두 외부에서 공급되는 전원에 의해 정보를 저장하기 때문에 휘발성 메모리에 해당
- 비휘발성 메모리(non-volatile memory)
  - 전원이 차단되어도 기록된 정보가 계속 유지
  - 자기 코어나 자기 디스크 메모리가 해당
  - 컴퓨터가 동작하는데 필요한 프로그램을 저장하는데 사용

# | Memory

## 접근 방법에 의한 분류

- RAM(Random Access Memory)
  - 접근 시간이 어느 위치나 동일하게 걸리는 메모리 형태
- SAM(Sequential Access Memory)
  - 원하는 위치에 도달하는데 일정한 시간이 경과되는 형태이므로 접근 시간은 위치에 따라서 다르다



# | Memory

## 기록 기능에 의한 분류

- RWM(Read and Write Memory)
  - 기록과 판독 두 가지를 모두 수행할 수 있는 메모리
- ROM(Read Only Memory)
  - 판독만 가능한 메모리

# Memory

## Memory의 종류: RAM

- SRAM(Static Random Access Memory)
  - 전원이 공급된 상태이면 기억된 정보를 계속 유지한다.
  - 기억 소자가 TTL Flip-Flop Logic으로 구성되어 있어 속도가 빠르다.
  - 집적도가 낮아서 용량이 적으며 소모 전력이 높다.
  - 주로 캐시 메모리에 사용된다.
  - 비트당 가격이 비싸다.
- DRAM(Dynamic Random Access Memory)
  - 전원이 공급된 상태에서 계속해서 재충전(refresh) 해 주어야만 기억된 정보를 유지한다.
  - 기억 소자가 CMOS로 구성되어 집적도가 매우 높다. 즉 기억 용량이 매우 크다
  - SRAM에 비해 접근속도가 느리며, SRAM에 비하여 약 5배정도 된다.
  - 주로 컴퓨터의 주 기억 장치로 사용된다.
  - 비트당 가격이 싸다

# | Memory

## Memory의 종류: RAM

- SDRAM(Synchronous DRAM)
  - Access is synchronized with an external clock
  - Since SDRAM moves data in time with system clock, CPU knows when data will be ready
  - Burst mode allows SDRAM to set up stream of data and fire it out in block
  - DDR-SDRAM sends data twice per clock cycle (leading & falling edge)

# Memory

## Memory의 종류: ROM

- Mask ROM(Read Only Memory)
  - Hard-Wired Logic 구조로서 공장에서 출하될 때에 이미 프로그램을 내장시켜서 내보내기 때문에 사용자가 프로그램을 구워 넣을 수가 없으며, 대량 생산일 때에 가격이 저렴
- PROM
  - PROM(Programmable Read Only Memory)은 Fuse-Burning Logic(실리콘 퓨즈)구조로 되어 있으며 단 한번만 사용자가 프로그램을 구워 넣을 수 있다.

# Memory

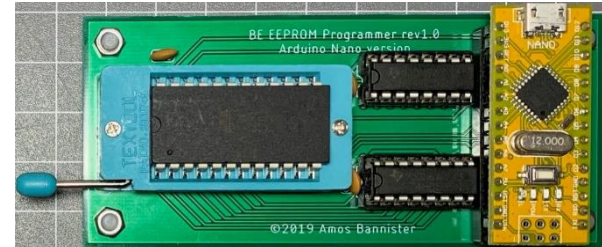
## Memory의 종류: ROM

- EPROM

- EPROM(Erasable Programmable Read Only Memory)는 소거 프로그램 가능 메모리로서 사용자가 프로그램을 여러 번 구워 넣을 수 있으며, 내부의 내용을 지울 때에는 자외선(ultraviolet)을 쏘이면 지워진다.

- EEPROM

- EEPROM(Electrically Erasable Programmable Read Only Memory)는 전기적인 충격으로 지우거나 프로그래밍 할 수 있는 기억장치이며 여러 번 사용할 수 있다.



# | Memory

## Memory의 종류: ROM

- FLASH

- EEPROM과 비교하여 값싸고 높은 용량의 메모리 구현 가능

- NOR FLASH

- erase/writes 속도가 느림
    - random reads 속도가 빠름
    - program code를 저장하고 수행하는데 유리
    - 집적도가 낮음

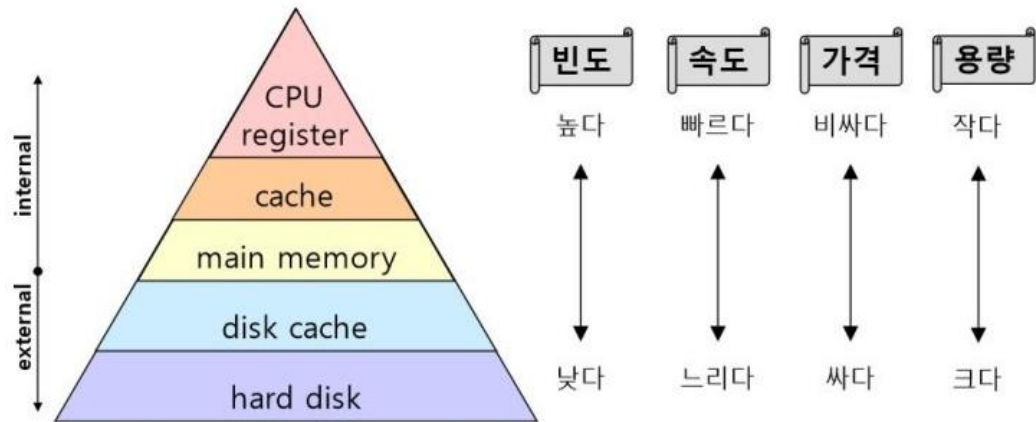
- NAND FLASH

- 높은 집적도를 통해 값이 저렴
    - erase/writes 속도가 빠름
    - block input/output access → random access 불가능
    - 데이터 저장용으로 사용

# Memory

## Memory Hierarchy

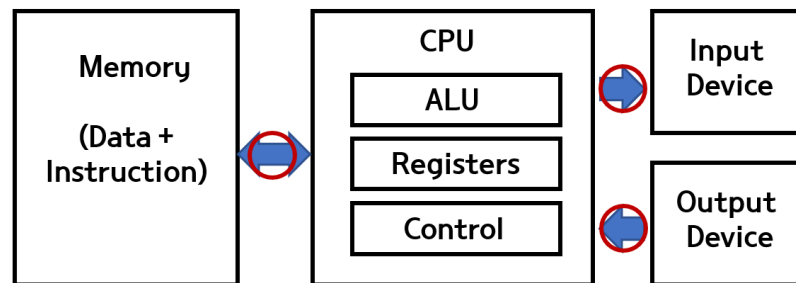
- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk Cache
- Disk
- Optical
- Tape



# Memory

## Von Neumann Bottleneck

- The separation between the CPU and memory  
→ the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory
- In most modern computers, throughput is much smaller than the rate at which the CPU can work  
→ the CPU is continuously forced to wait for needed data to be transferred to or from memory
- The performance problem is reduced by a cache between the CPU and the main memory

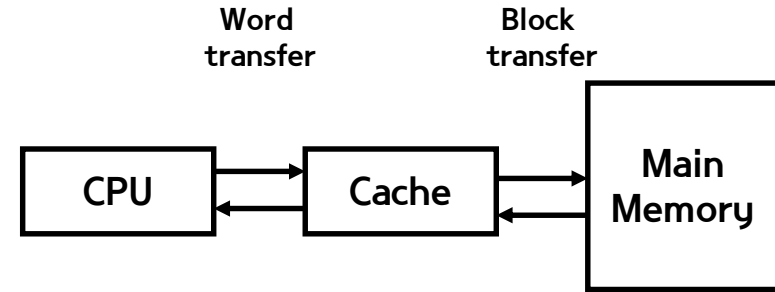




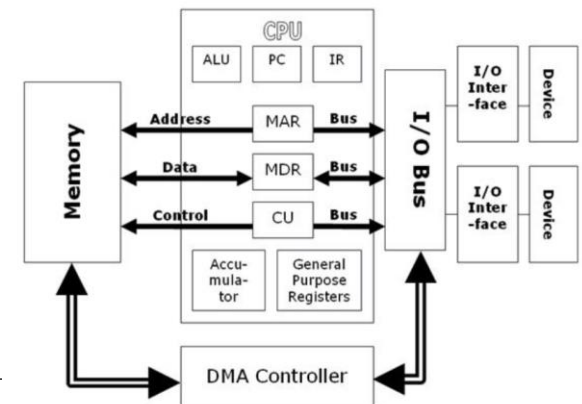
# Memory

## Cache

- Fast CPU  $\leftarrow \rightarrow$  slow main memory
- Small amount of fast memory
- Between normal main memory and CPU
- External or internal
- Data and instruction cache  $\leftarrow$  Harvard architecture
- Intel Core i7
  - 32KB L1 instruction and 32KB L1 data cache per core
  - 256KB L2 cache (combined instruction and data) per core
  - 8MB L3 (combined instruction and data) shared by all cores

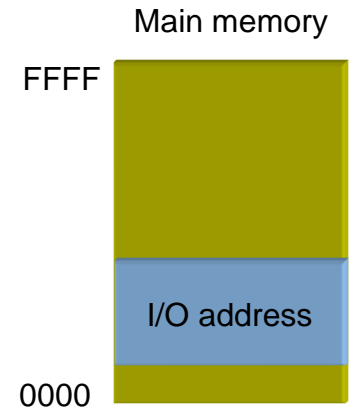


- Programmed IO
  - CPU has direct control over IO
    - Sensing status
    - Read/write commands
    - Transferring data
  - CPU waits for IO module to complete operation
  - Wastes CPU time
- DMA (direct memory access)
  - Memory/IO accesses are done independently of CPU
  - IO to IO to memory, memory to memory
  - A DMA controller (HW) is required
  - Fast block transfer
  - No burden to CPU



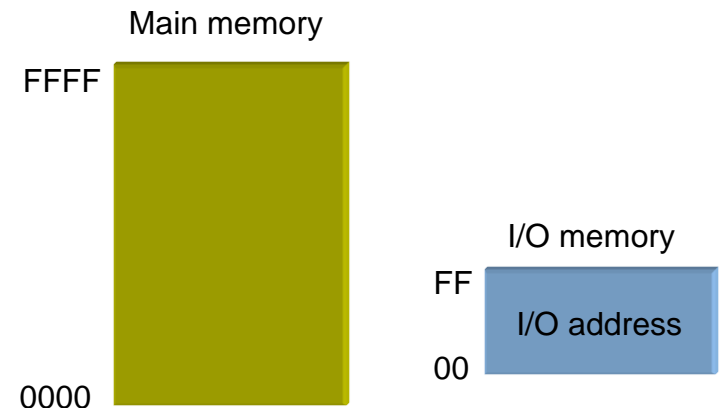
- **Memory-mapped I/O**

- Main memory 내부에 I/O address 영역을 두는 방식
- 회로가 간단
- Main memory 영역 감소



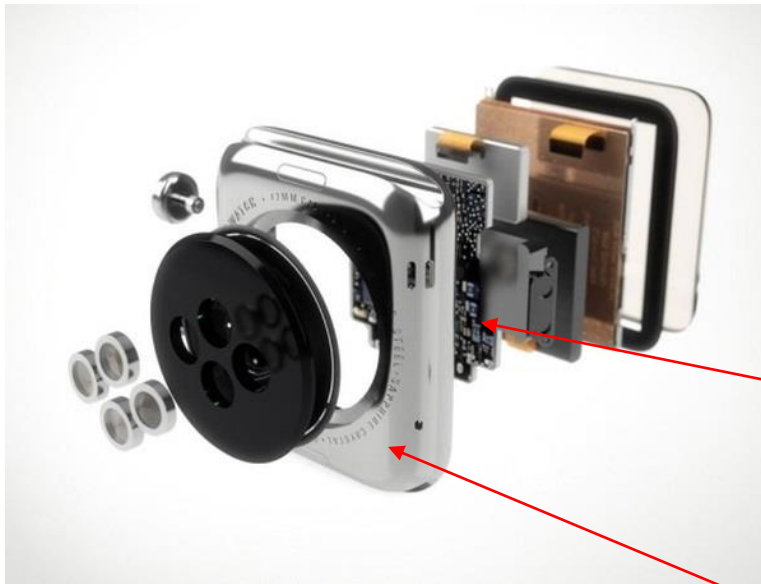
- **I/O-mapped I/O**

- Main memory 영역과 I/O address 영역을 분리 시켜서 설계된 방식
- 회로가 복잡
- I/O를 위한 명령어 필요
- Main memory 영역을 모두 사용 가능



# Embedded System

- Apple Watch



Exploded View

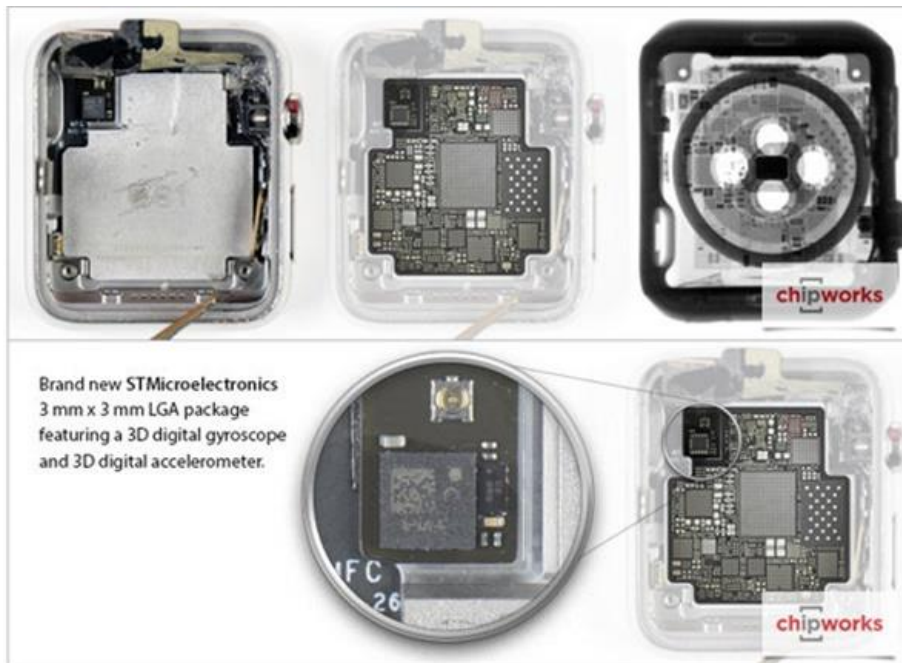


회로부

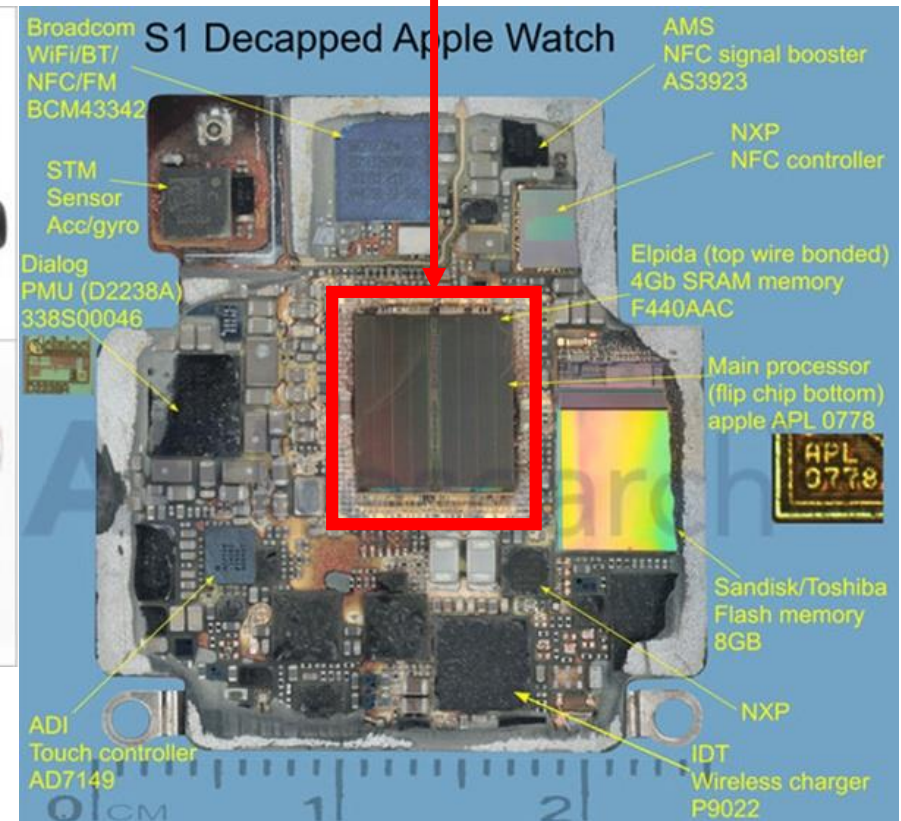
기구부

# Embedded System

- Apple Watch



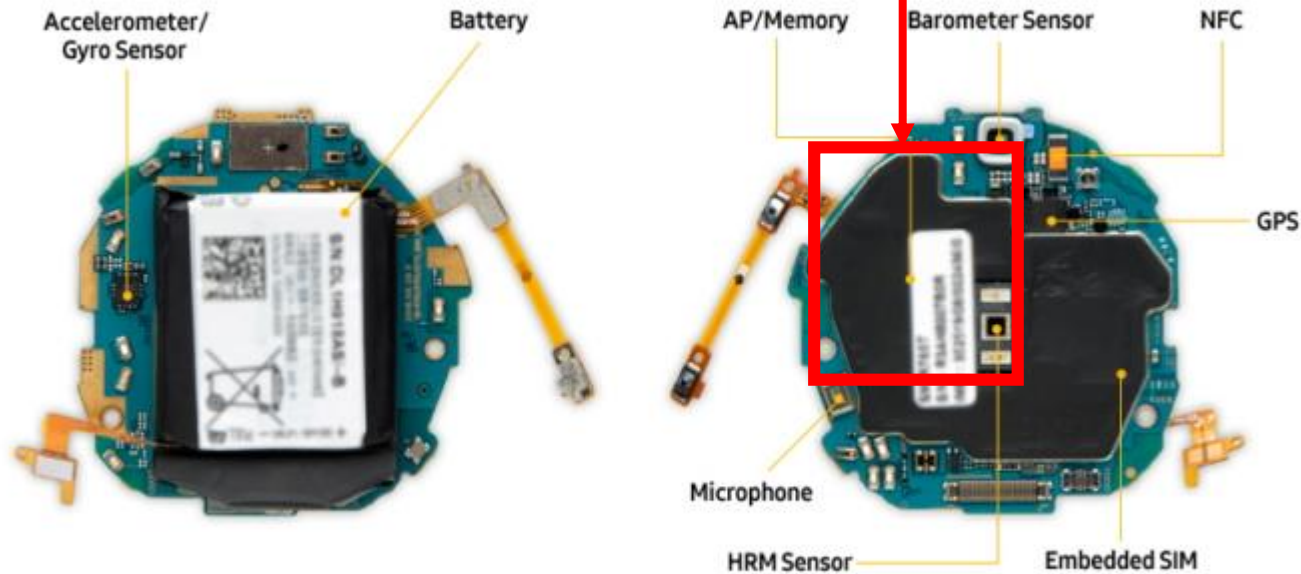
## Microprocessor



# Embedded System

- 삼성 기어 S3

Microprocessor  
AP : application processor



Gear S3 Mainboard (front / back)

# ARM 프로세서

- 영국의 Acorn 社에서 1983년부터 디자인
  - ARM1(1985년), ARM2(1986년) 개발
  - Acorn RISC Machine(ARM)
- Advanced RISC Machines Ltd. 설립(1990년)
  - ARM Ltd. 사명 변경(1998년), Advanced RISC Machine(ARM)
  - ARM7TDMI 이후 모바일 제품 및 게임기 시장에서 성공
  - ARM11 이후 Cortex까지 개발되고 있음
  - 팹리스(Fabless) 반도체 기업으로 ARM 코어를 IP(지식재산권)형태로 판매

- Cortex processor family
  - Cortex-A
    - 'A' : application
    - Cortex 계열 중 가장 성능이 우수
    - 스마트폰 메인 프로세서(엑시노스, 스냅드래곤 등)
  - Cortex-R
    - 'R' : real-time
    - 실시간 처리와 관련된 응용 프로그램에 사용
  - Cortex-M
    - 'M' : micro-controller
    - 저가형 응용 프로그램(기존 16bit or 낮은 성능의 32bit 프로세서 영역)



# Cortex-M 프로세서

- Cortex-M0
  - 저전력, 낮은 가격
- Cortex-M7
  - Cortex-M 계열 프로세서 중 가장 성능이 우수
- Cortex-M3/Cortex-M4
  - Cortex-M 계열 프로세서 중 널리 사용
  - 저가격 및 저전력의 32bit 프로세서
  - 하바드 구조(Cortex-M0는 폰노이만 구조)
  - Thumb-2 명령어 지원(높은 코드 밀도, 16bit/32bit 명령어를 섞어서 실행가능)
  - Memory mapped I/O 방식
  - 부동소수점 연산 유닛 (Floating-point unit, FPU) 옵션 (Cortex-M4F)

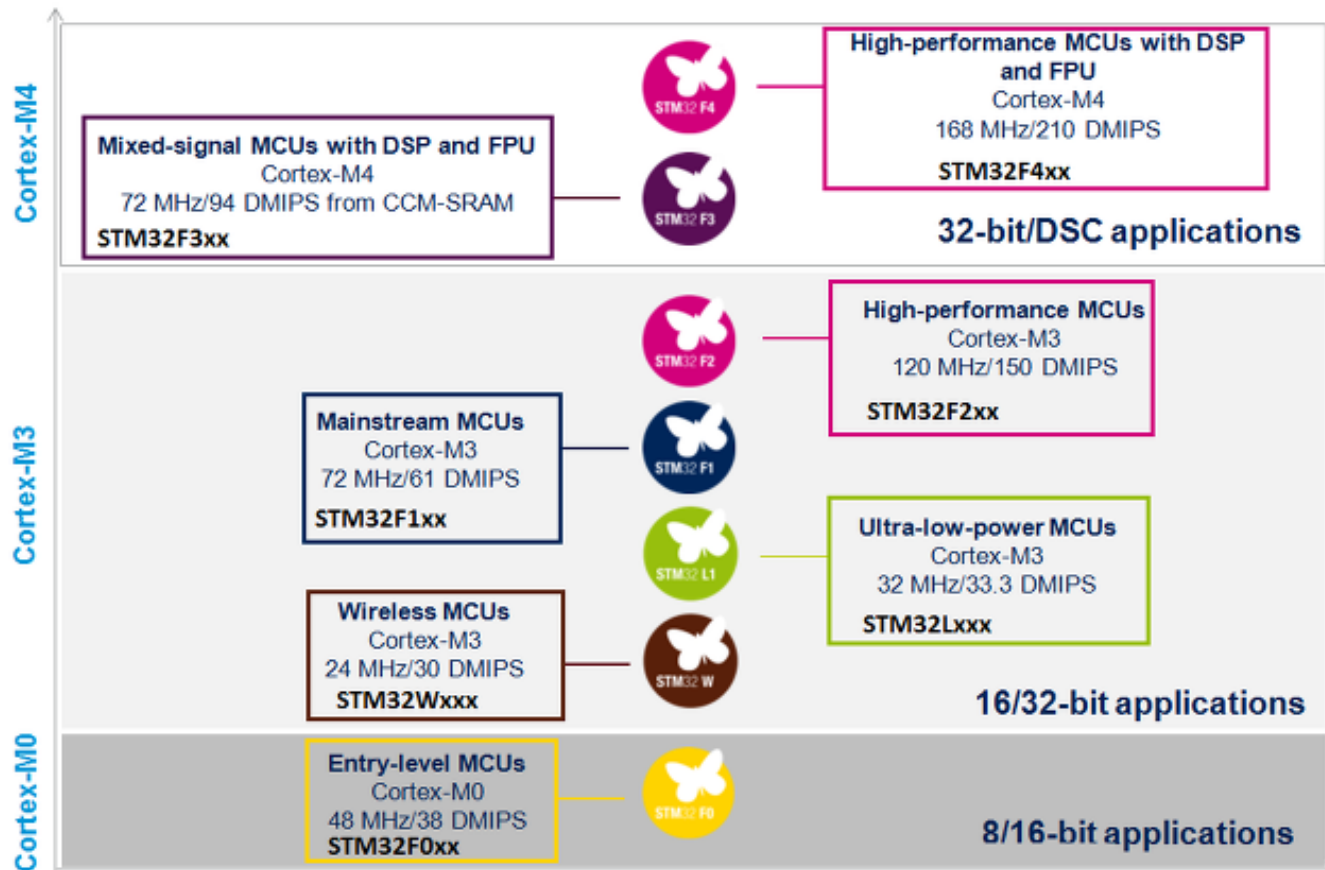
# Cortex-M MCUs

- STM32 a comprehensive platform



# Cortex-M MCUs

- STM32 a comprehensive platform



# STM32F405의 개요

- ARM사에서 개발된 RISC(Reduced Instruction Set Computer) 구조의 저전력 CMOS 32-Bit 프로세서인 Cortex-M4 Core를 내장한 ST사의 고성능 마이크로컨트롤러
- 코어 명은 ARMv7이며 제품 군의 이름은 Cortex(코어 텍스)로 명명
- STM32F405는 STM32 제품 군들 중에서 상위의 모델에 해당하며, 최대 168MHz 속도로 동작
- Flash 메모리와 SRAM을 제공하며 이외에 다양 주변장치를 제공
- STM32 프로세서는 다양한 응용 분야에 적용되어 사용 중
- 임베디드 프로세서 코어 설계 회사로 유명한 ARM 사에서는 ARM7/ARM9/ARM11으로 이어지는 32bit 임베디드 프로세서를 제공하고 있는데, 2006년 새로운 아키텍처를 기반으로 하는 제품 군을 추가로 발표

# | 응용 분야

- 마이크로프로세서 응용 전 분야
- 8bit 마이컴 이상의 고성능 응용분야
- CAN, USB등 고성능 통신 응용분야
- Motor Control(DC, BLDC, Step Motor)
- Small Webserver



# I STM32F405의 특징

- 사양
  - 고성능 저전력 Cortex-M4 32bit Micro-controller
    - 향상된 Harvard 아키텍처(1.25DMIPS/MHz@168MHz)
  - 비 휘발성 프로그램과 데이터 메모리
  - 1M Byte 내부 프로그램 가능한 ST-LINK Flash memory
  - 선택적인 Boot code section (used In-System Programming by On-chip Boot Program)
  - ST-LINK (In System Programming)를 통해 어플리케이션 영역과 부트 영역에 있어 F/W 다운로드 가능
  - 192+4K Byte 내부 SRAM
- 내장 메모리의 Programing과 On-Chip Debug를 위한 JTAG (IEEE Standard 1149.1) 지원

# I STM32F405의 특징

- 12개의 16비트 타이머/카운터(각각 4개의 IC/OC/PWM)
- 2개의 32비트 타이머/카운터(각각 4개의 IC/OC/PWM)
- 2개의 워치독 타이머
- 1개의 systick timer (24bit down counter)
- 분리된 오실레이터에 의한 Real Time Count
- Output Compare Modulator
- 세 개의 12bit ADC, 24 채널
- 두 개의 12bit DAC

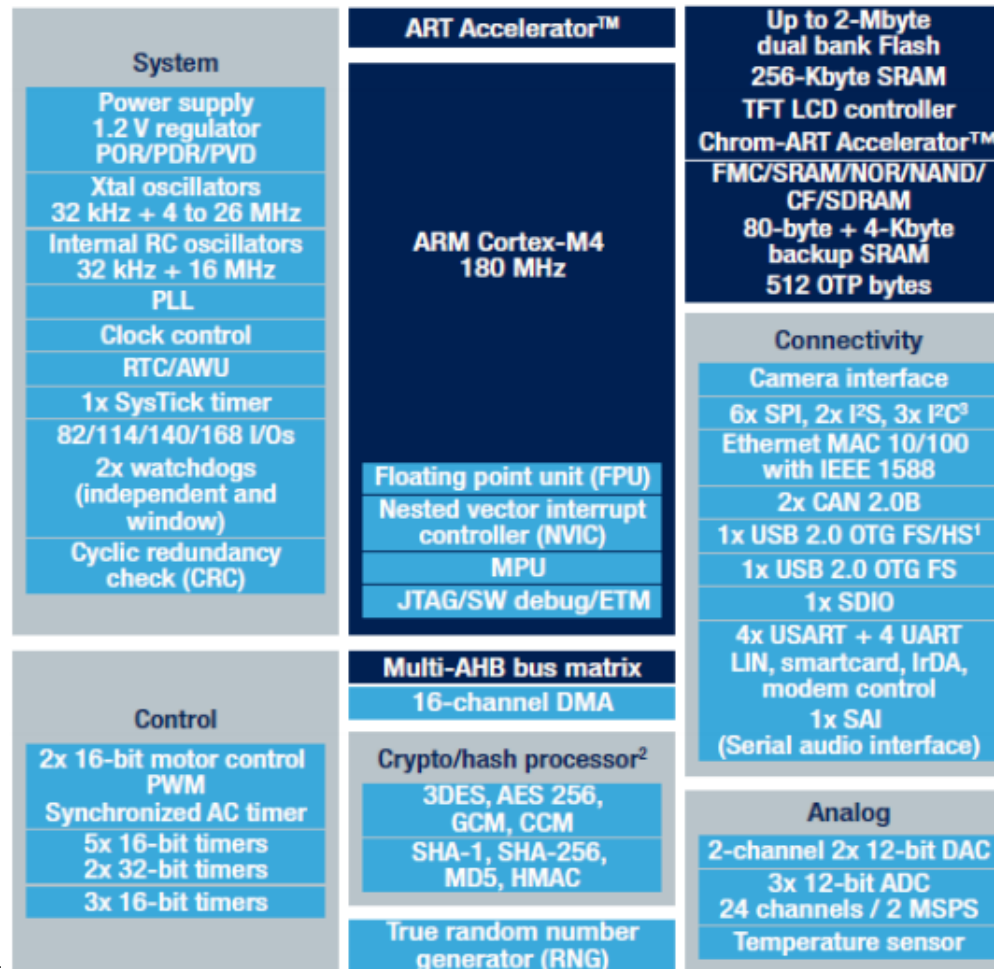
# STM32F405의 특징

- 세 개의 Two-wire Serial 인터페이스
- 네 개의 시리얼 USART/ 두 개의 시리얼 UART
- 세 개의 Master/Slave SPI 시리얼 인터페이스(42Mbit/s)
- 프로그램 가능한 워치 독(Watchdog) 타이머
- 두 개의 CAN 인터페이스
- 한 개의 USB 2.0 full speed 인터페이스
- 한개의 10/100 Ethernet MAC
- 8~14bit 병렬 카메라 인터페이스(54 Mbytes/s)



# STM32F405의 특징

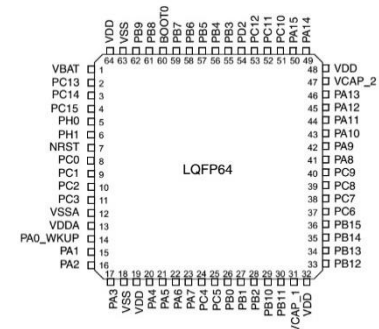
## • STM32F405내부구조



# STM32F405의 구조

- STM32F405 외형과 핀 기능

- VBAT (핀1) : 백업 전원 입력 핀으로 파워 세이브 모드 시 이를 통해 전원을 공급 받아 최소 전원으로 동작이 가능하도록 한다.
- RESET (핀7) : 입력 단자로 '0'레벨이 입력되면 리셋 되어 PC(Program Counter)는 일반적으로 0번지를 가리키고 0번지부터 프로그램이 시작된다. 리셋 시 대부분의 레지스터는 초기화된다.
- OSC\_IN, OSC\_OUT (핀5,6) : 발진 용 증폭기 입력 및 출력 단자.
- VDD (핀19,32,48,64) : 전원 입력 단자.



# STM32F405의 구조

- STM32F405 외형과 핀 기능

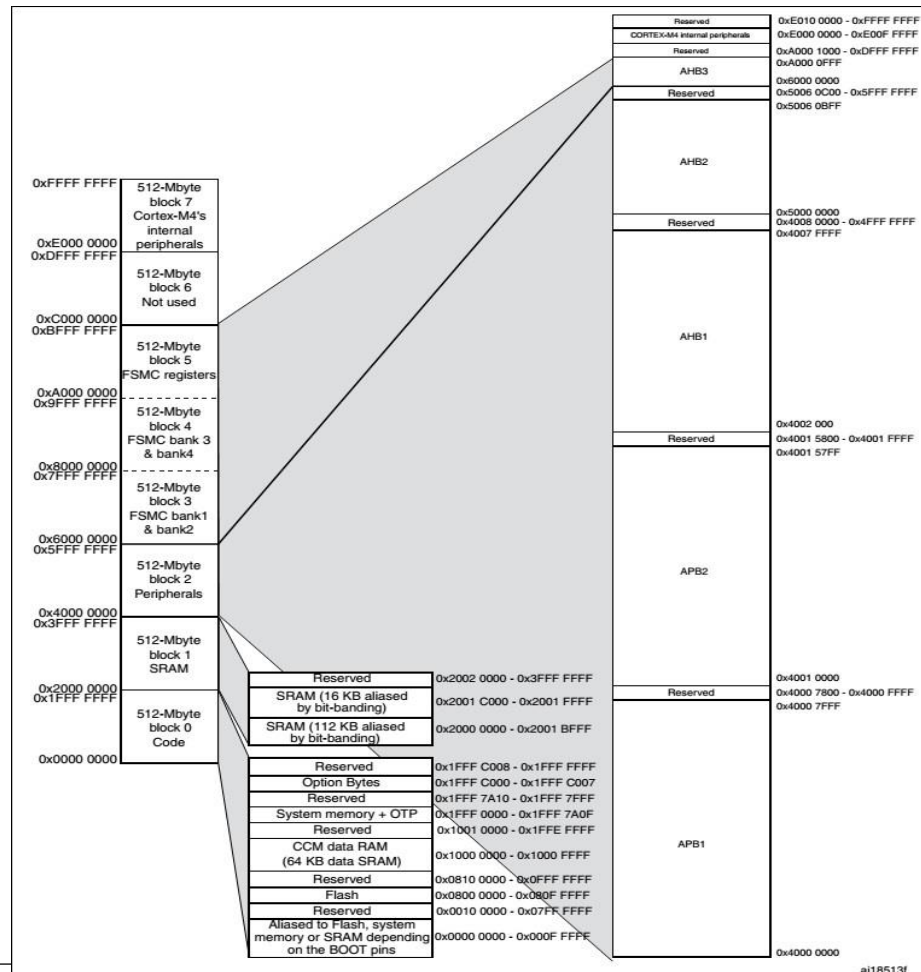
- GND (핀12,18, 63) : 그라운드 입력 단자.
- VDDA (핀13) : AD변환기 및 포트 F에 대한 공급 전압
- 포트A (PA15~PA0) : 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자. ADC, UART, SPI, I2C, USB, CAN, ETHERNET, DCMI, JTAG용 단자로 사용된다.
- 포트B (PB15~PB0) : 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자. ADC, UART, SPI, I2C, USB, CAN, ETHERNET, FSMC, DCMI, JTAG용 단자로 사용된다.
- 포트C (PC15~PC0) : 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자. ADC, UART, SPI, I2C, USB, ETHERNET, SDIO, DCMI용 단자로 사용된다.
- 포트D (PD2) : 내부 풀업, 풀다운 저항이 있는 1비트 양방향 입출력 단자. UART, I2C, USB, DCMI,용 단자로 사용
- 포트H (PH0~PH1) : 내부 풀업, 풀다운 저항이 있는 2비트 양방향 입출력 단자. 외부 클럭 입력용 단자로 사용

# I STM32F405의 구조

- STM32F405의 메모리는 프로그램 메모리와 데이터 메모리가 분리된 하이브리드 구조
- 프로그램 코드를 저장하고 실행시키기 위해 필요한 메모리를 프로그램 메모리라 하며, STM32F405의 경우 1M 바이트의 크기로 JTAG 프로그램이 가능한 내부 플래시 메모리가 이에 해당된다.
- 반면 프로그램 코드가 실행되는 도중 발생하는 데이터를 전원이 켜 있는 동안 일시적으로 저장하기 위해 마련된 읽고(Read) 쓰기(Write)가 가능한 메모리를 데이터 메모리라 하며 192+4k바이트의 SRAM이 이에 해당된다(4K 바이트는 Backup SRAM).
- 각종 Peripheral을 위한 레지스터 및 포트 제어를 위한 어드레스 영역을 제공한다.

## STM32F405의 구조

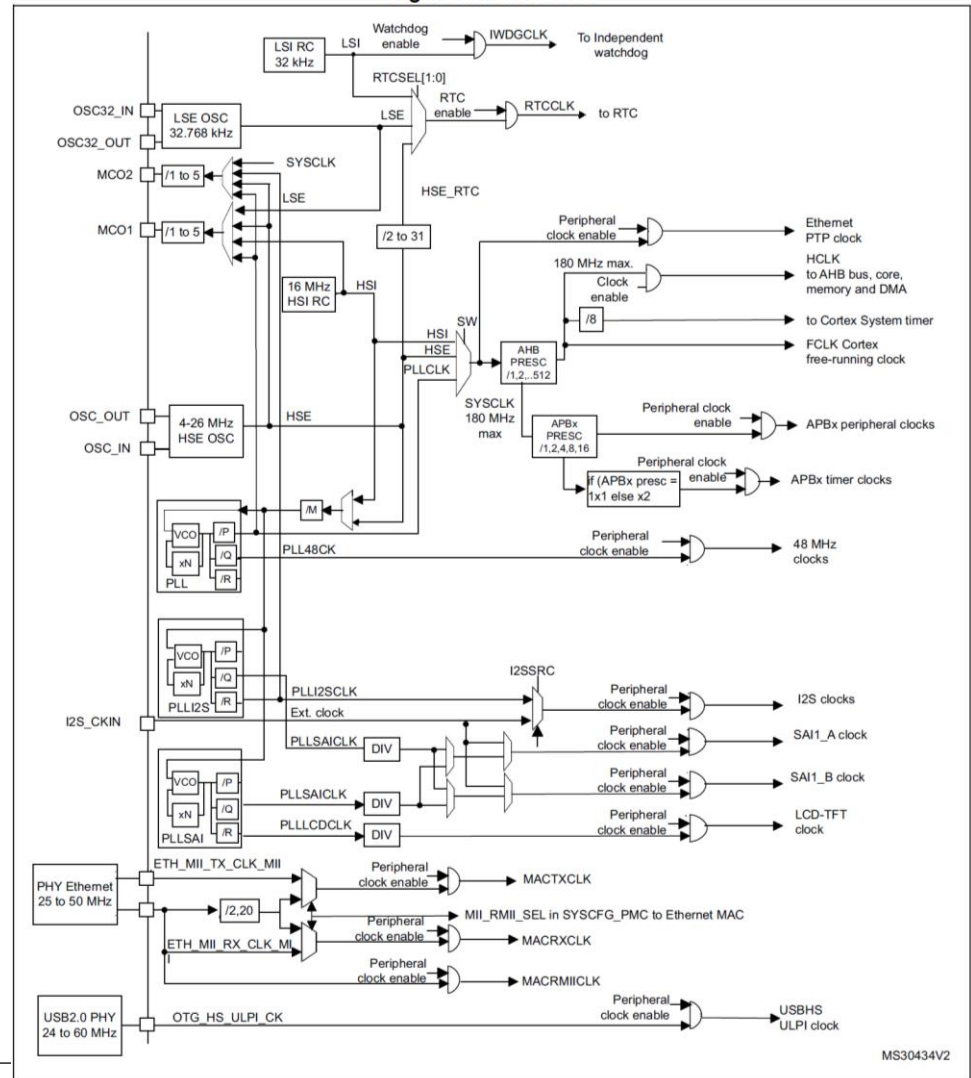
- STM32F405의 프로그램 메모리와 데이터 메모리 맵 구조



# STM32F405의 구조

## • STM32F405의 클럭

Figure 16. Clock tree



MS30434V2

# 과제

## 실습환경 구축

- SPL (Standard Peripheral Libraries) 다운로드
  - <https://www.st.com/en/embedded-software/stsw-stm32065.html>
  - 이메일 인증 필요
  - 일단 다운만 받아 놓기

# 과제

## 실습환경 구축

- IAR Embedded Workbench for Arm 설치
  - <https://www.iar.com/kr/products/architectures/arm/iar-embedded-workbench-for-arm/>
  - 무료 평가판 설치
  - 이메일 인증 필요, 제공 키 값 입력 (추후 교육용으로 갱신 예정)



**Thank you.**

