Chapter 1장-3장

1부 마이크로프로세서란?

담 당 : 이인수교수

강의 개요

1. 교재 및 참고 문헌

- 임베디드 시스템을 위한 마이크로프로세서 기초에서 활용까지, 권우현
- 참고자료 : PIC16F84의 기초+ a, 이희문, 성안당
  Microcontroller PIC 16C84/71, 차영배, 다다미디어
  PIC16F877/874와 그 응용, 진달복, 양서각

2. 강의 진형 방법 및 활용 매체

- 이론 강의 및 실험 실습 : 2인 1조를 원칙으로 함

3. 평가 방법

- 출　　석 10%
- 보고서 및 과제수행 20%

  : 예비보고서(실험내용,예비문제포함) 및 결과보고서(실험결과 및 고찰, Home Work 포함)
- 중간고사 40% , 기말고사 30%(Term project)

4. 연락처

- 연구실 : IT-1호관 711호, Tel. 950-7843,
- 이메일 : insoolee@knu.ac.kr

마이크로프로세서의 기초

▣ 무엇을 배우나

- 기본적인 용어
- 마이크로 프로세서란?
- 마이크로프로세서의 내부구조

마이크로프로세서(Microprocessor) 란?

: 컴퓨터의 산술논리연산기, 레지스터, 프로그램 카운터, 명령디코더, 제어회로 등의 연산장치(ALU)와 제어장치(CU)를 1개의 작은 실리콘 칩에 모아놓은 처리장치.
주기억장치에 저장되어 있는 명령을 해석하고 실행하는 기능을 한다

→컴퓨터에서 명령을 수행하고 데이터를 처리하는 **중앙처리장치(CPU)**를 집적회로의 칩 형태로 만든 것

Digital & Analog

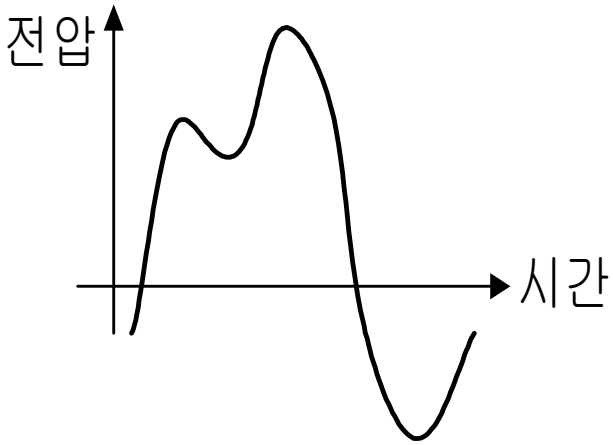
ㅁ아날로그 신호(Analog Signal)

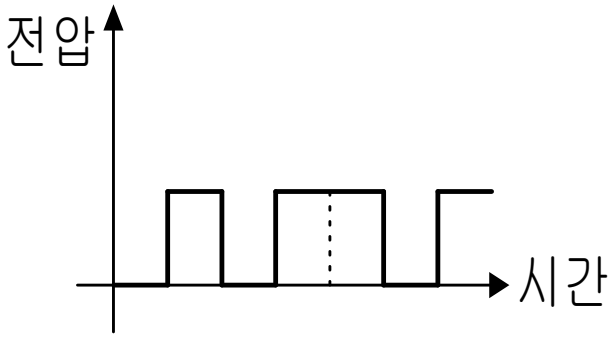
■자연계에서 일어나는 물리적인 양은 시간에 따라 연속적으로 변화.

■온도, 습도, 소리, 빛 등은 시간에 따라 연속적인 값을 갖는다.

ㅁ 디지털 신호 (Digital Signal)

■ 분명히 구별되는 두 레벨의 신호값 만을 갖는다.





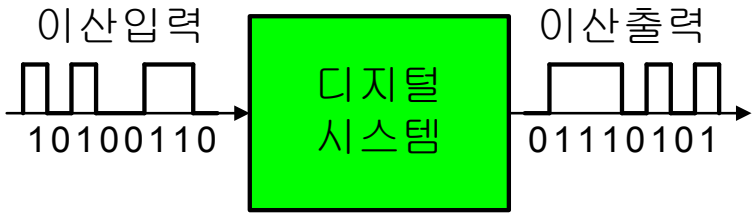
Digital & Analog System

◘ 디지털 시스템

■ 이산적인 정보를 가공하고 처리해서 최종 목적으로 하는 정보를 출력하는 모든 형태의 장치

◘ 아날로그 시스템

■연속적인 정보를 입력 받아 처리해서 연속적인 형태의 정보를 출력하는 시스템



◘ 디지털 시스템의 장점

■ 디지털 시스템은 내·외부 잡음에 강함

■ 디지털 시스템은 설계하기가 용이

■ 디지털 시스템은 프로그래밍으로 전체 시스템을 제어할 수 있어서 규격이나 사양의 변경에 쉽게 대응할 수 있어서 기능 구현의 유연성을 높일 수 있고 개발기간을 단축시킬 수 있음.

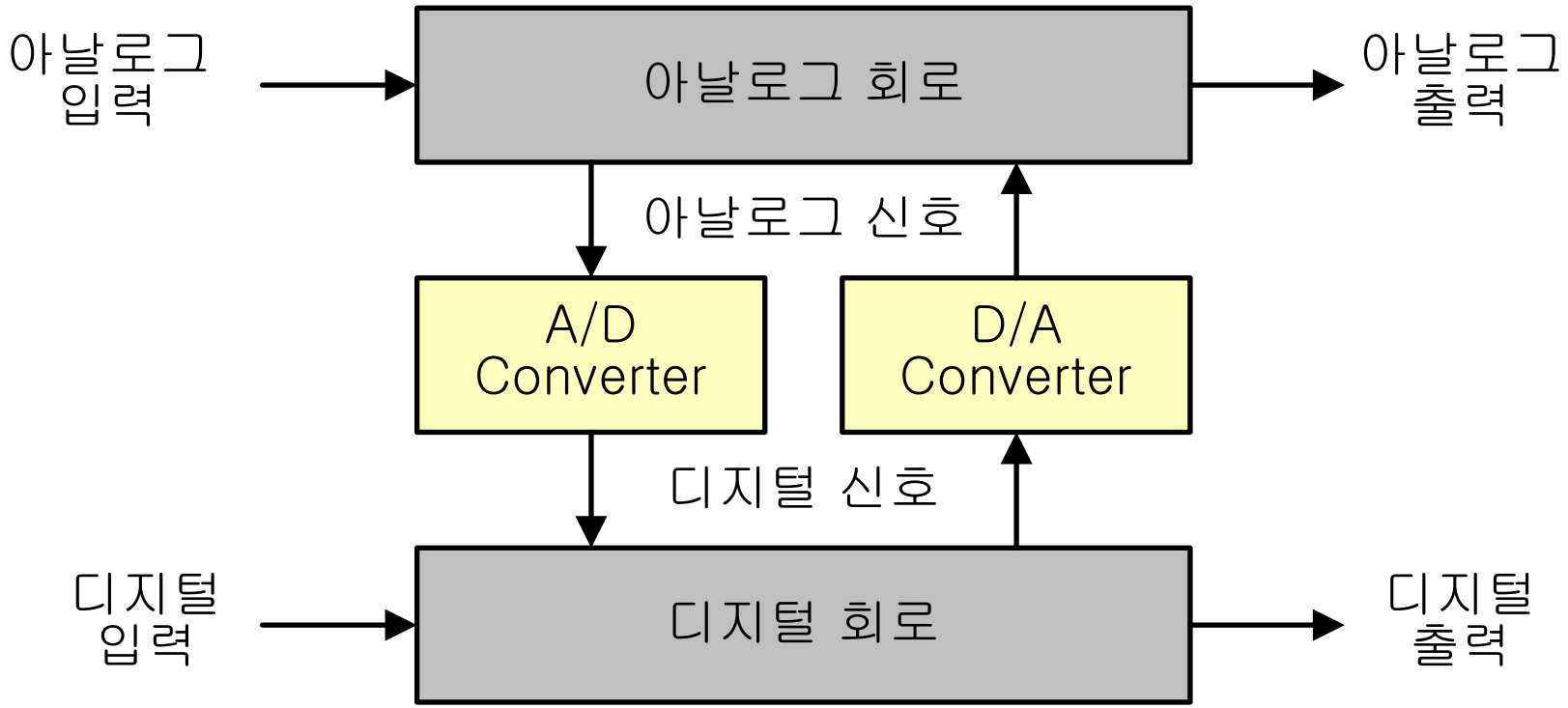
■ 디지털 시스템에서는 정보를 저장하거나 가공하기가 용이.

■ 디지털 시스템에서는 정보처리의 정확성과 정밀도를 높일 수 있으며, 아날로그 시스템으로는 다루기 어려운 비선형 처리나 다중화 처리 등도 가능.

■ 디지털 시스템은 전체 시스템 구성을 소형화, 저가격화로 할 수 있음.

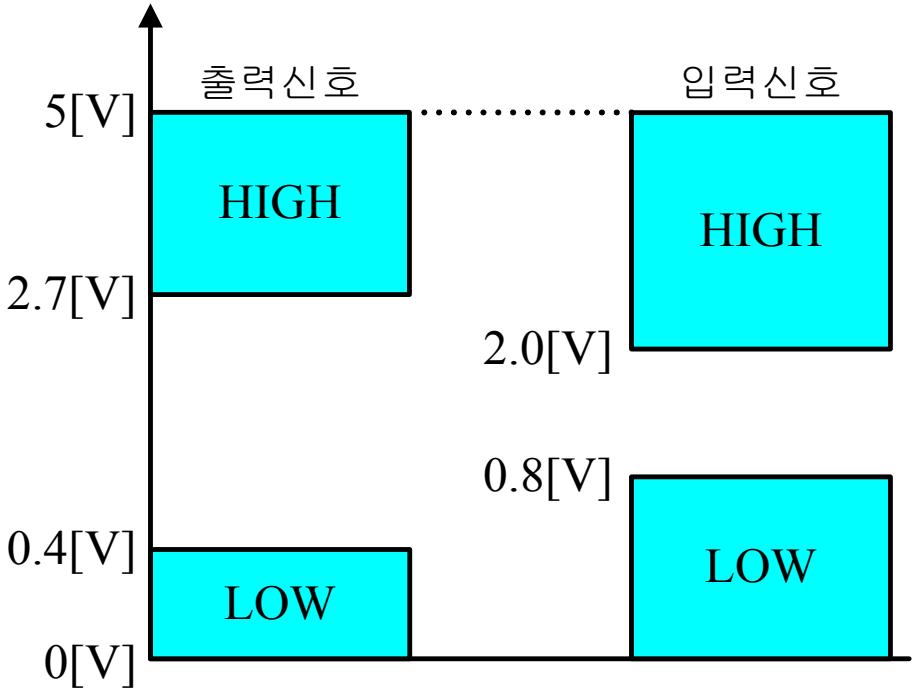
➢ **디지털 시스템의 많은 장점으로 인해 기존 아날로그 시스템이나 새로운 시스템의 대부분은 디지털 시스템으로 구성**

Digital & Analog System의 연결



Digital 정보의 표현

- 디지털 정보의 전압 범위
  - 디지털 정보를 표현하기 “0”과 “1”의 2진수 체계(binary system)사용



기본 용어

■ 디지털 정보와 디지털 IC

➢ 사칙연산 식에 숫자를 대입하여 생활에 적용

$Z = A \times B$

A : 버스의 최대 탑승 인원

B : 버스 대수

Z : 최대 승차인원

- (사례) 40인승 버스 8대에 승차할 수 있는 최대 인원은 320

➢ 논리연산 식에 논리 값을 대입하여 생활에 적용

$Z = X \text{ AND } Y$

X : 방범창살의 파손 여부 (파손 : 1, 정상 : 0)

Y : 유리창 열림 여부 (열림 : 1, 닫힘 : 0)

Z : 침입 감지 여부 (감지 : 1, 감지되지 않음 : 0)

- (사례) 방범창살이 정상이고 유리창이 열려 있다면, 침입은 감지되지 않음

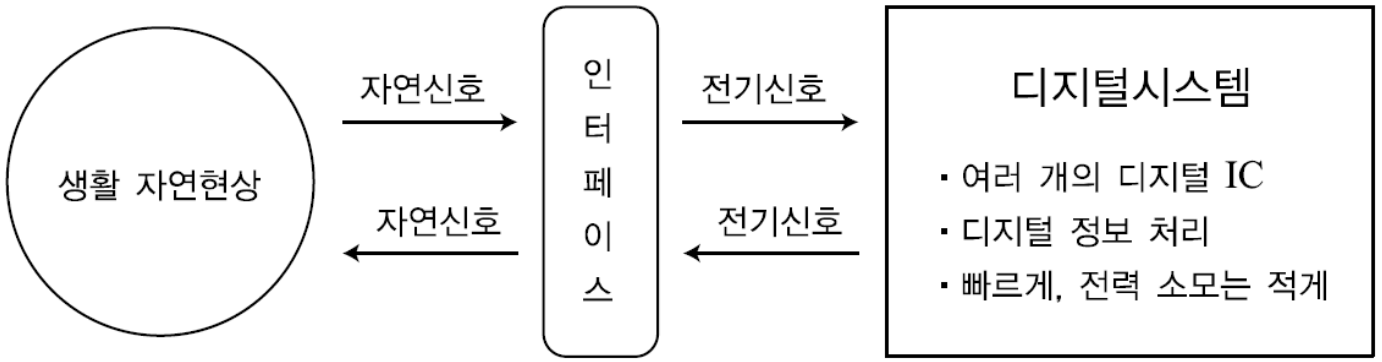
기본 용어

➢ 디지털 정보 표현

- 숫자 : 2진수 체계를 갖는 한정된 비트의 디지털값으로 변환된 정보
- 문자 : 한정된 비트 크기의 2진화된 코드로 변환된 정보
- 영상, 음성 : 샘플링 단위를 중심으로 해상도에 의해 제한된 비트 크기의 코딩 기법으로 변환된 정보
- 논리 : 1과 0의 2진 논리 정보(참일 때 1, 거짓일 때 0)

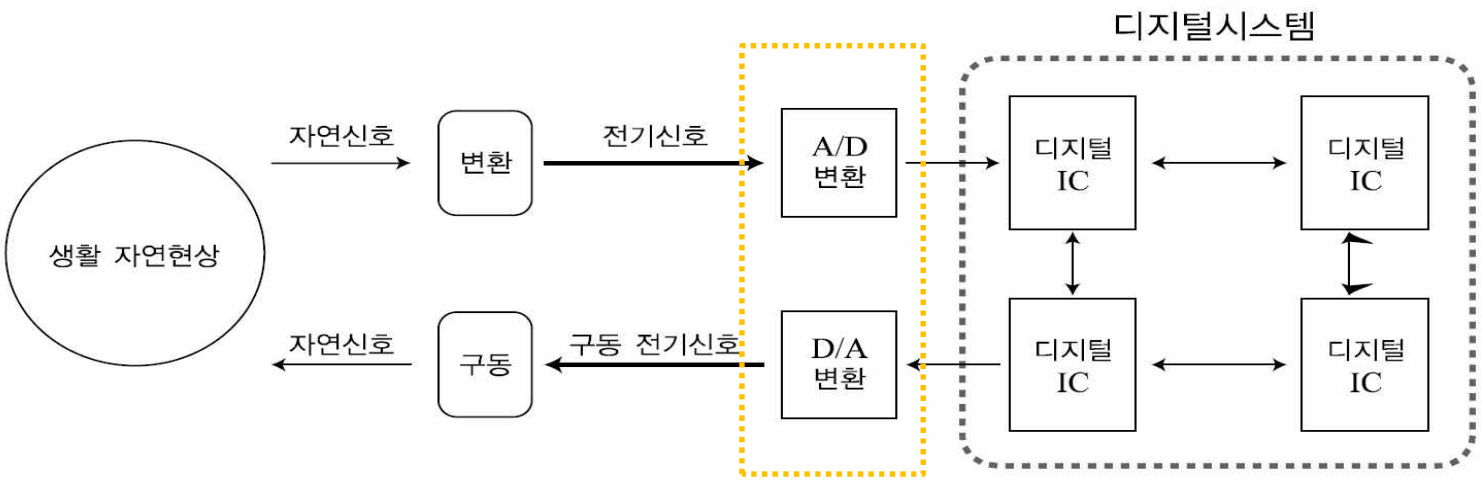
➢ 디지털 IC

- 문자, 영상, 음성, 논리 등을 디지털 정보화하고, 디지털 IC 내부에서 전기신호로 빠르게 처리 (열, 바람, 습도 등→ 센서이용 자연신호를 전기신호로 변환)



기본 용어

■ 정보처리용 디지털 IC의 I/O 장치 인터페이스 (예 : 히터사용온도제어, 온도표시)



➢ **디지털시스템의 디지털 IC 사이는 빠른 디지털 정보 연산과 정보 전달**

- 고속의 저전력 신호 사용

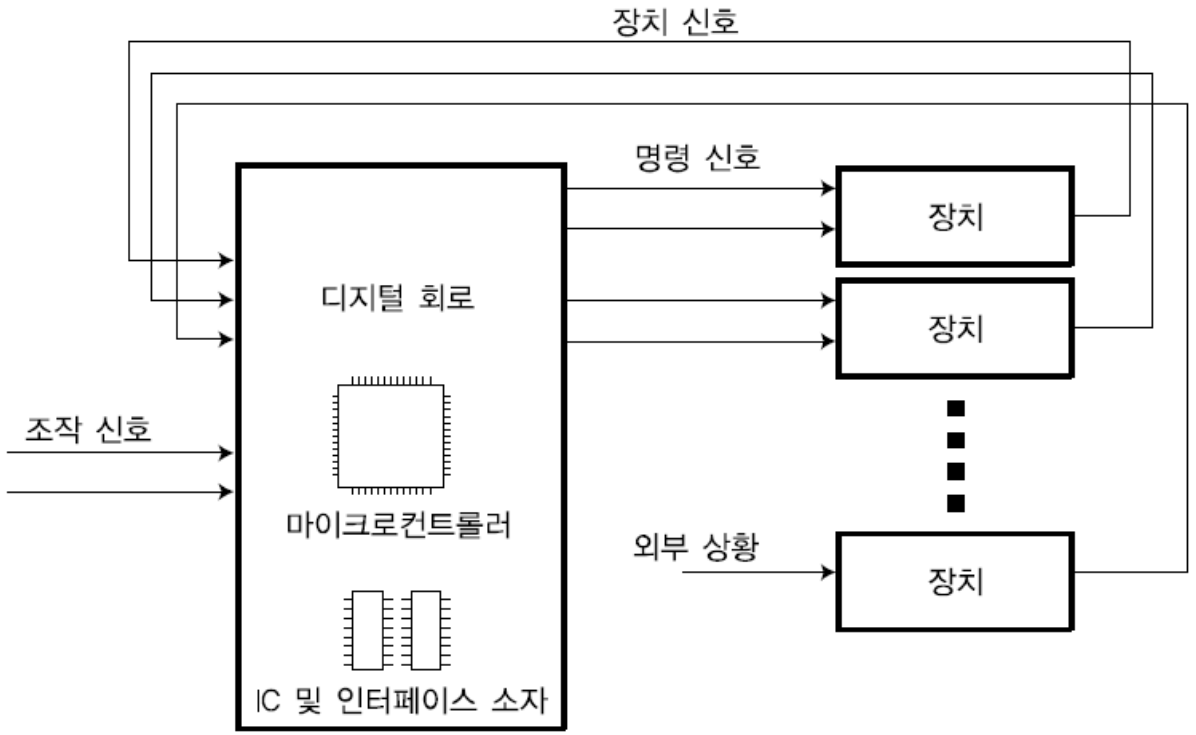
➢ **디지털시스템의 계산 결과를 자연신호로 변환 사용**

- 연산 결과 정보를 구동 전기신호로 생활 자연현상에 적용
- 자연현상에 영향을 주기 위한 충분한 전기에너지로 변환하여 장치 구동

마이크로컨트롤러와 디지털시스템

■ 디지털 시스템

- 아날로그시스템과 상대되는 용어
- 마이크로컨트롤러가 포함된 디지털 회로로 사용자의 목적에 맞게 장치를 제어하기 위한 시스템



■ 디지털 시스템 동작

❶ 제어 프로그램은 장치 제어를 위해 명령 신호를 출력

❷ 장치는 명령 신호를 입력받아 명령에 대한 동작을 수행

❸ 마이크로컨트롤러를 포함한 디지털 회로는 아래 신호를 입력

- 명령 신호에 따라 장치가 작동되면서 출력되는 장치 신호
- 외부 상황을 측정하는 장치에서 출력되는 장치 신호
- 사람이 시스템 동작을 위해 인가하는 신호

❹ 마이크로컨트롤러는 입력 신호를 받아 제어 프로그램의 진행 상태를 변경하면서 장치 제어를 위해 필요한 출력을 결정

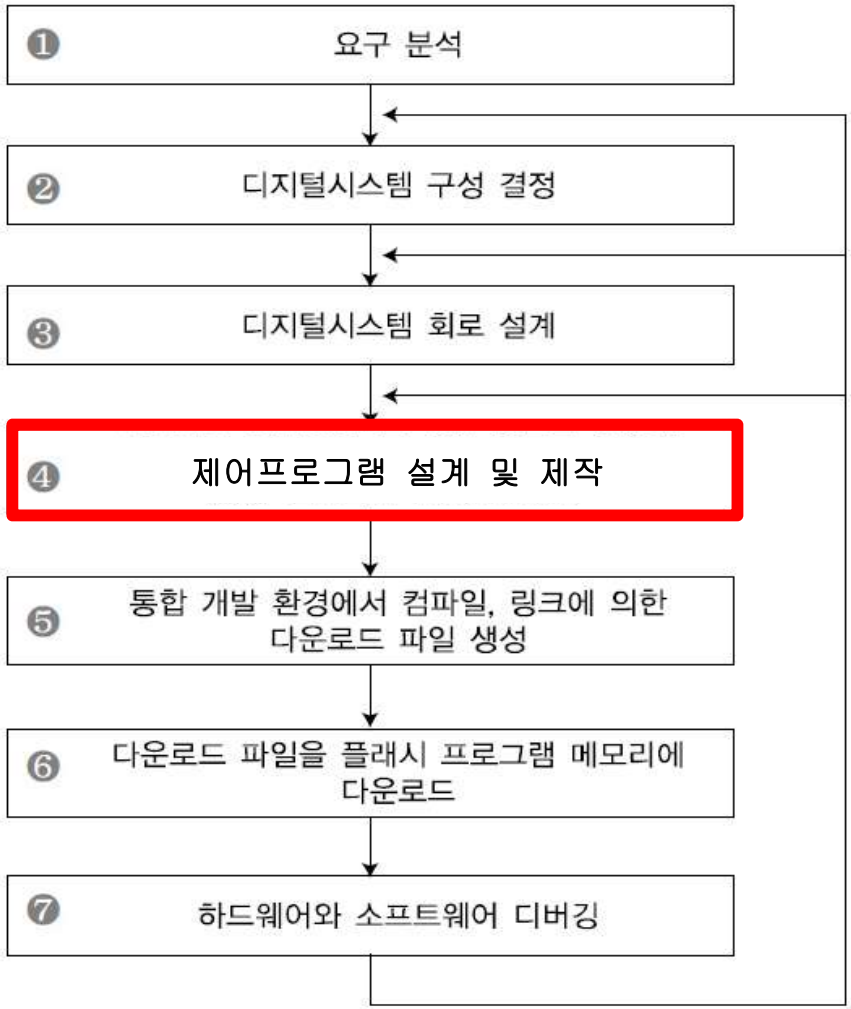
■ 장치

- **제품이거나 전자 소자 또는 부품으로 만든 장치**
- **디지털 회로에서 명령 신호가 인가되면 작동**

■ 디지털 회로

- **마이크로컨트롤러를 포함한 IC 및 인터페이스 소자들이 연결된 회로**
- **제어 프로그램이 수행되면서 만들어진 명령 신호는 장치를 작동시킴**
- **마이크로컨트롤러**
  - 제어 프로그램을 수행하면서 위의 작업 단계를 수행시키는 핵심 제어부
- **IC 및 인터페이스 소자**
  - 외부 장치를 인터페이스하려면, IC를 추가하거나 인터페이스 회로를 설계하여 장치와 연결

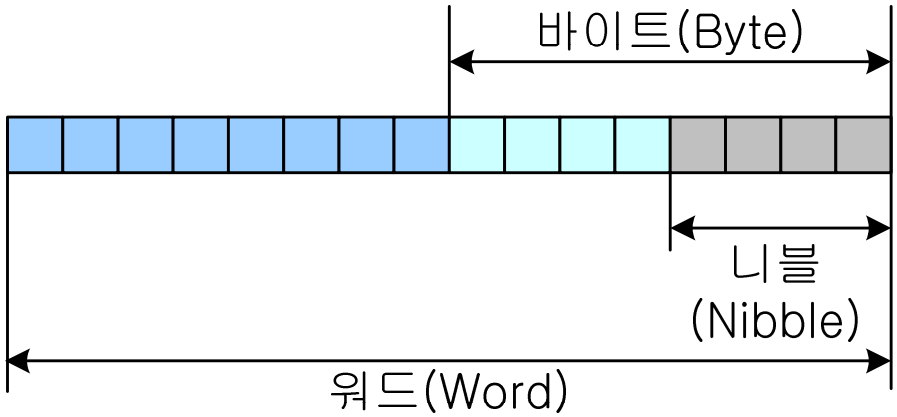
■ PIC16F876을 이용한 디지털시스템 개발 과정



기본 용어

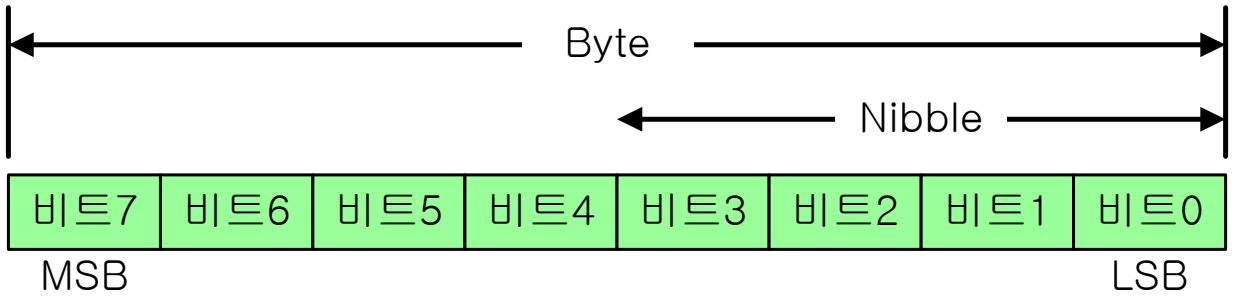
▣ 기본 용어 1

1. 2진수(Binary)
   - High level(5V) : 1, Low level(0V) : 0
2. 비트(Bit)
   - 2 진수의 1 자리수
3. 니블(Nibble)
   - 2 진수의 4 자리수
4. 바이트(Byte)
   - 2 진수의 8 자리수
5. 워드(Word)
   - 2 진수의 16 자리수
6. 더블워드(Double Word)
   - 2 진수의 32 자리수



Digital 정보 단위의 표현

- 1 nibble = 4 bit
- 1 byte = 8 bit
- 1 byte = 1 character
- 1 문자 표현 : 영어→1 byte, 한글 → 2 byte
- 1 word : 특정 CPU에서 취급하는 명령어나 데이터의 길이에 해당하는 비트 수



2^{10} byte = 1024 byte = 1 Kbyte

2^{20} byte = 1024 Kbyte = 1 Mbyte

2^{30} byte = 1024 Mbyte = 1 Gbyte

2^{40} byte = 1024 Gbyte = 1 Tbyte

기본 용어

▣ 기본 용어1

7. 16 진수(Hexadecimal)
   - 2진수 4개의 비트 즉 니블 ☞ 16진수 한 자리
   - 0000 ☞'0', 0001 ☞ '1', 0010 ☞ '2' … 1010 ☞ 'A', 1011 ☞ 'B', … 1111 ☞ 'F'
   - ABH ☞ 0ABH, 1BH ☞ 1BH
8. MSB (Most Significant Bit) / LSB (Least Significant Bit)

기본 용어

▣ 기본 용어2

1. **레지스터(Register)**
   - 기억장치의 일종으로서 다양한 용도로 사용됨
2. **버스(Bus)**
   - CPU와 메모리, I/O 소자 등과의 연결에 사용되는 신호선
   - 데이터 버스(양방향), 어드레스 버스(단방향), 컨트롤 버스(단방향)
3. **메모리(Memory)**
   - 롬(ROM)/램(RAM)
     - ☞ 종류 및 특징 조사
4. **포트(Port) : 외부와의 입,출력 기능을 수행하는 물리적인 부분**
   - I/O (Input, Output Device), I/O Port

참고

■ 메모리

➢ 플래시 프로그램 메모리

- 비휘발성 메모리이기 때문에 전원이 없어도 데이터를 계속 유지
- **8K x 14** words of FLASH Program(플래시 프로그램 메모리)가 내장
- 쓰기, 지우기를 반복할 수 있기 때문에 프로그램을 직접 변경하면서 ISP(In System Programming) 방법으로 개발

-→ ISP(In System Programming)로 재 프로그램 가능한 플래시 메모리 내장

- 프로그램 저장, 10,000회 정도 쓰고 읽기가능

➢ 데이터 메모리(SRAM : Static RAM)

- 전원이 공급되는 동안은 저장되어 있는 정보가 유지됨
- 프로그램에서 선언한 변수와 스택을 위해 읽고, 쓰기를 빠르게 수행할 수 있는 주 메모리
- **368 x 8 bytes** of Data Memory (SRAM)
- 휘발성 메모리이므로, 전원이 없으면 데이터 소멸

참고

➢ **EEPROM (Electrically Erasable Programmable ROM)**

- 프로그램 실행 중 생성된 데이터를 전원 없이도 유지시키기 위해서 EEPROM 사용, 256 x 8 bytes of EEPROM Data Memory
- 프로그램이 수행되는 동안에 데이터를 기록하여 활용
- 기록할 때 SRAM, 플래시 프로그램 메모리보다 시간이 오래 걸림

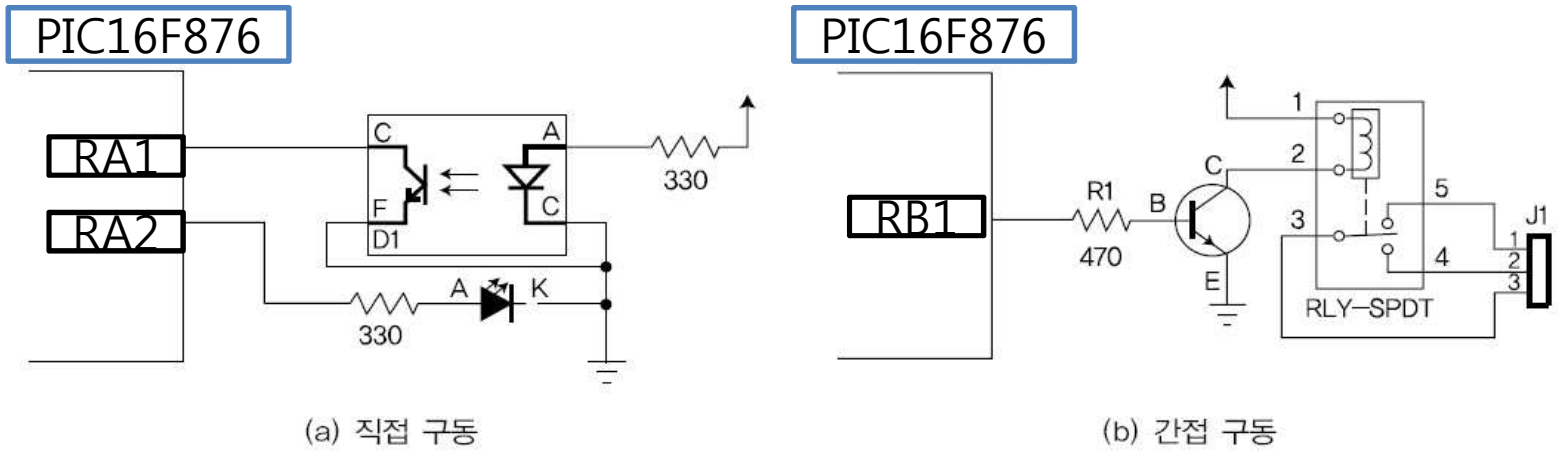
■ 양방향 I/O 포트

➢ **디지털 전압레벨과 논리값의 대응**

- **최대 22개의 I/O 핀을 사용하여** 마이크로컨트롤러 외부의 디지털 전압레벨과 내부의 논리값을 대응
  - ✓ 핀의 신호 방향을 출력으로 설정하고,
  - ✓ PORTx(x는 A~C) 레지스터에 0 또는 1에 해당하는 논리값을 기록하면,
  - ✓ 핀 Pxn(x는 A~C, n은 0~2)은 디지털 전압레벨로 변환되어 신호 출력

➢ 직접구동, 간접구동

- 간단한 스위칭 소자를 구동할 수 있는 전류로 공급 또는 흡수할 수 있음
- 단독으로 구동하기 어려운 장치는 스위칭 소자를 추가하여 구동할 수 있음



[I/O 핀을 이용한 직접 구동과 간접 구동]

참고 : 포토인터럽터 인터페이스

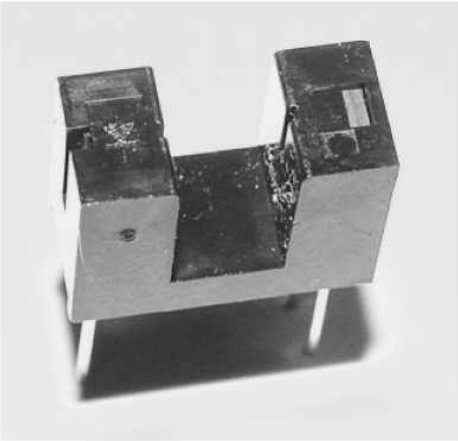
■ 사전 지식

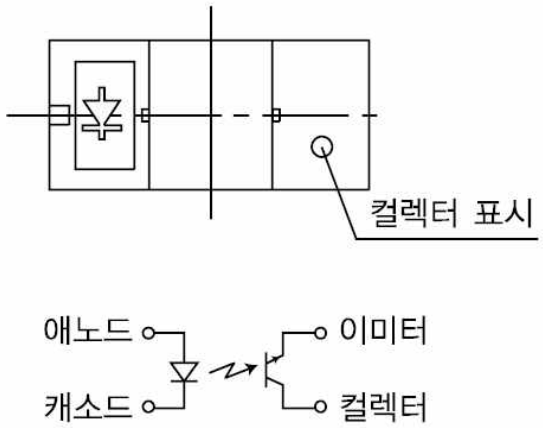
➢ **포토인터럽터는 포토다이오드와 포토트랜지스터를 일체형으로 만든 부품**

➢ **포토인터럽터 SG-207[(주)광전자 제품]을 중심으로 주요 규격 검토**

- 약 940[nm]의 적외선이 포토트랜지스터의 베이스로 투과
- 베이스 단자에 전압이 생성되고, 이미터로 충분한 순방향 전류가 흐름
- 컬렉터와 이미터 사이의 스위치를 ON시킴

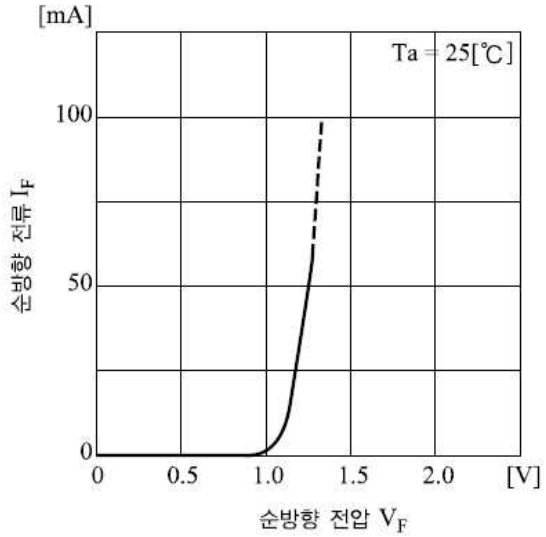
➢ **포토다이오드와 포토트랜지스터 간격에 적외선을 차단시켜 센서 동작**



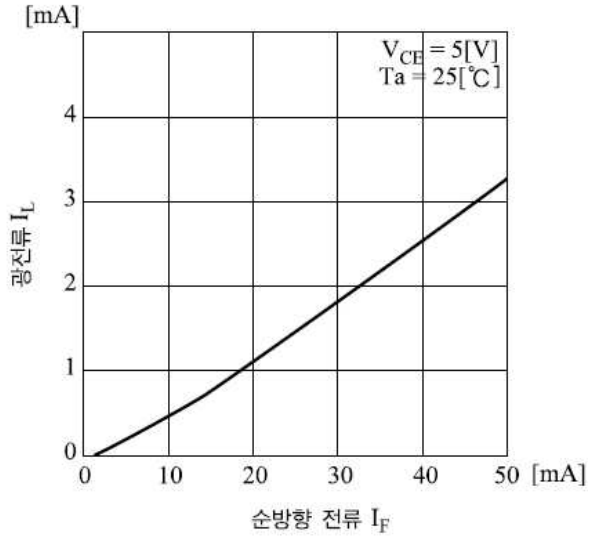


참고 : 포토인터럽터 인터페이스

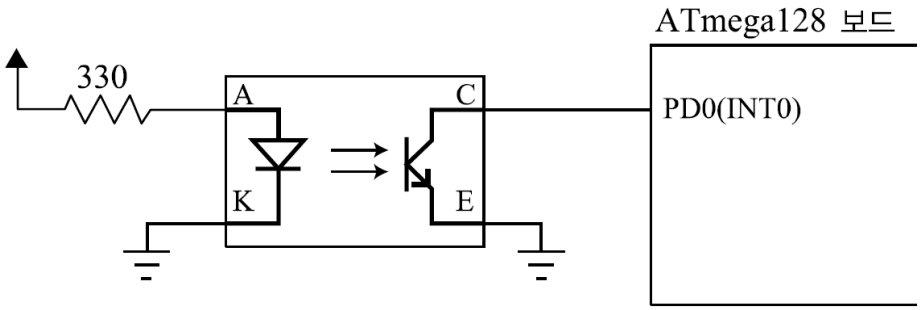
➢ 특성곡선



(a) 다이오드 순방향 전압과 전류의 특성 곡선



(b) 다이오드 전류와 트랜지스터 광전류와의 특성 곡선



- ATmega128 내부 풀업저항 20[kΩ] 이상 가정

참고 : 포토인터럽터 인터페이스

➢ **적외선 관통 포토트랜지스터에 도달되면 ON, LOW 디지털 전압레벨**

- 다이오드 순방향 전류 계산

$$\frac{5-1.2}{330} \simeq 12[\mathrm{mA}]$$

- 12[mA]에 의한 적외선 관통 예측 광전류 $0.5[\mathrm{mA}]$
- 트랜지스터 ON

$$0.5[\mathrm{mA}] \times 20[\mathrm{k\Omega}] + V_{CE(Sat)} > 5[\mathrm{V}]$$

→ 트랜지스터의 포화동작을 만족시킴

- 입력 핀 PD0는 전압 VCE(Sat)이 되어 LOW 디지털 전압레벨
- 핀에서 인식되는 논리값은 '0'

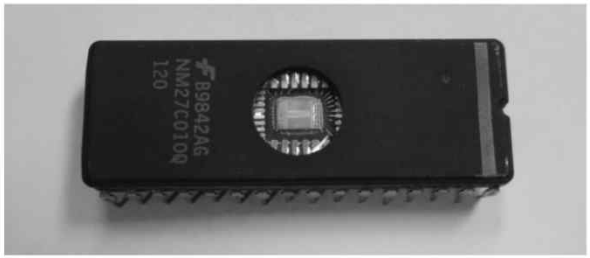
➢ **적외선이 차단되면 포토트랜지스터 OFF, HIGH 디지털 전압레벨**

- 적외선이 차단되었을 때, 광전류 I_L은 0[mA]
- 따라서, 핀에 인가되는 전압은 V_{CC}이 되어 HIGH 디지털 전압레벨
- 핀에서 인식되는 논리값은 '1'

참고: ISP(In System Programming)

■ EPROM

- 예전 개발자들은 주로 EPROM을 사용하며 자외선으로 데이터 소거 및 프로그램
- 자외선으로 데이터 소거
- 많은 시행 착오를 거쳐서 완성



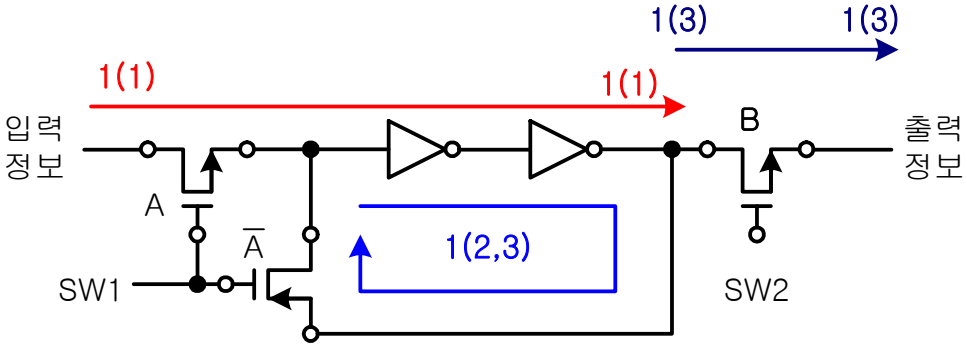
[EPROM]

■ ISP (In System Programming)

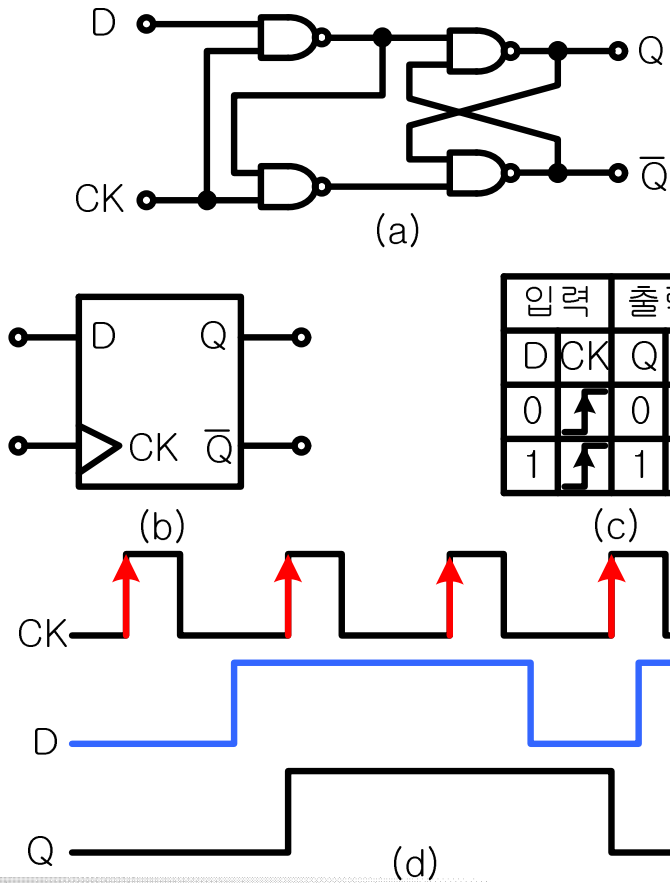
- 마이크로컨트롤러를 장착한 PCB에 직접 장착하여 시스템을 개발
- 프로그램 개발과 하드웨어 디버깅 과정을 한꺼번에 처리
- 내장된 플래시 메모리에 실행 코드를 직접 다운로드
  - 개발에서 생산까지 소요되는 시간을 단축

기본 용어-Register

- 레지스터 : 데이터를 일시보존 및 데이터를 다른곳으로 옮기는 일을 수행
  - 1bit 저장 장치의 구조
  - D Flip-Flop
  - D Latch

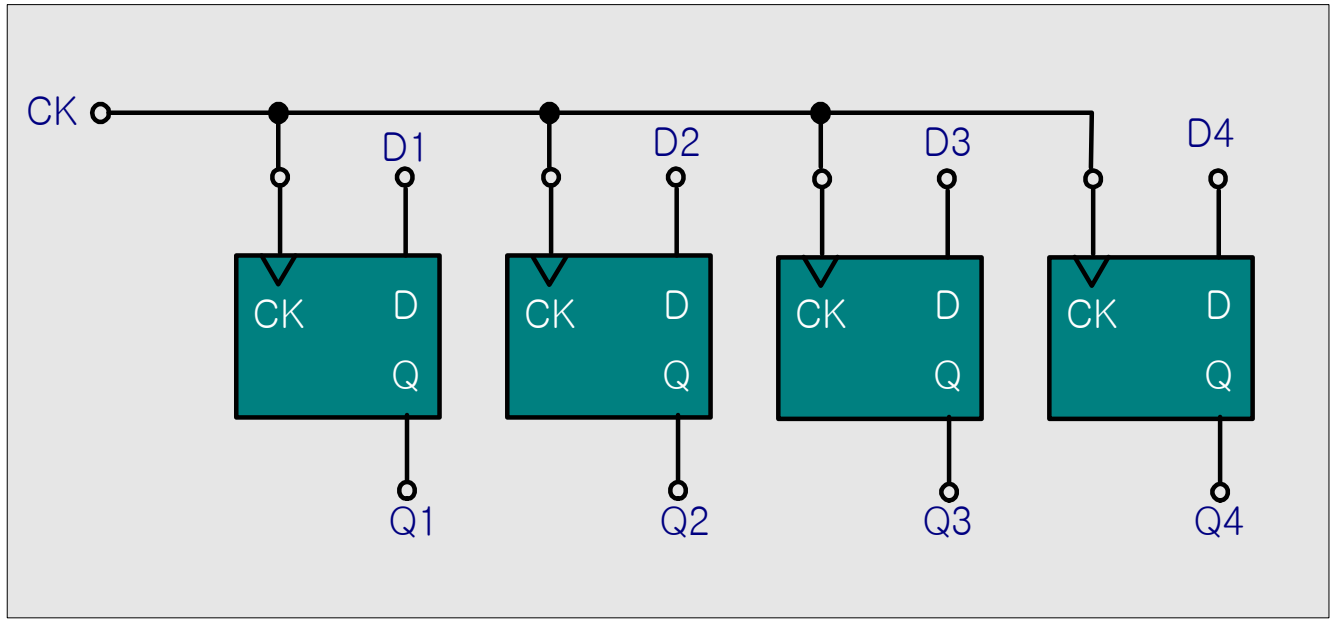


[Feedback을 이용한 1bit 저장 장치]

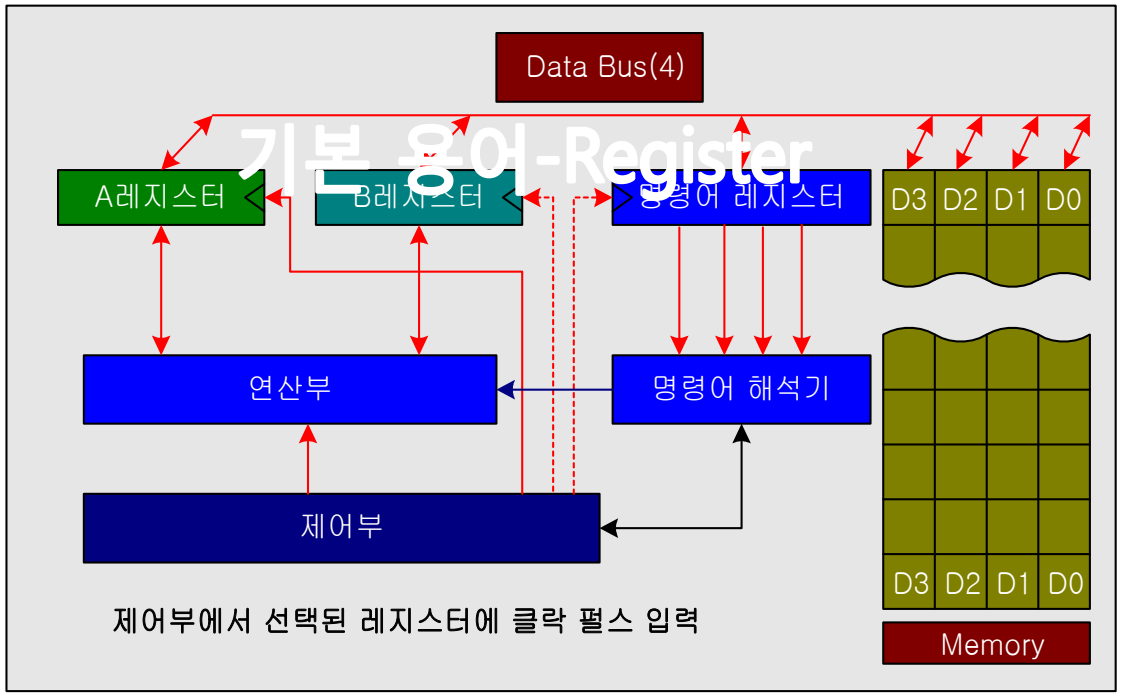


기본 용어-Register

- 4bits 레지스터
  - D Flip-Flop 으로 구성된 기억 소자



◘ 레지스터 선택

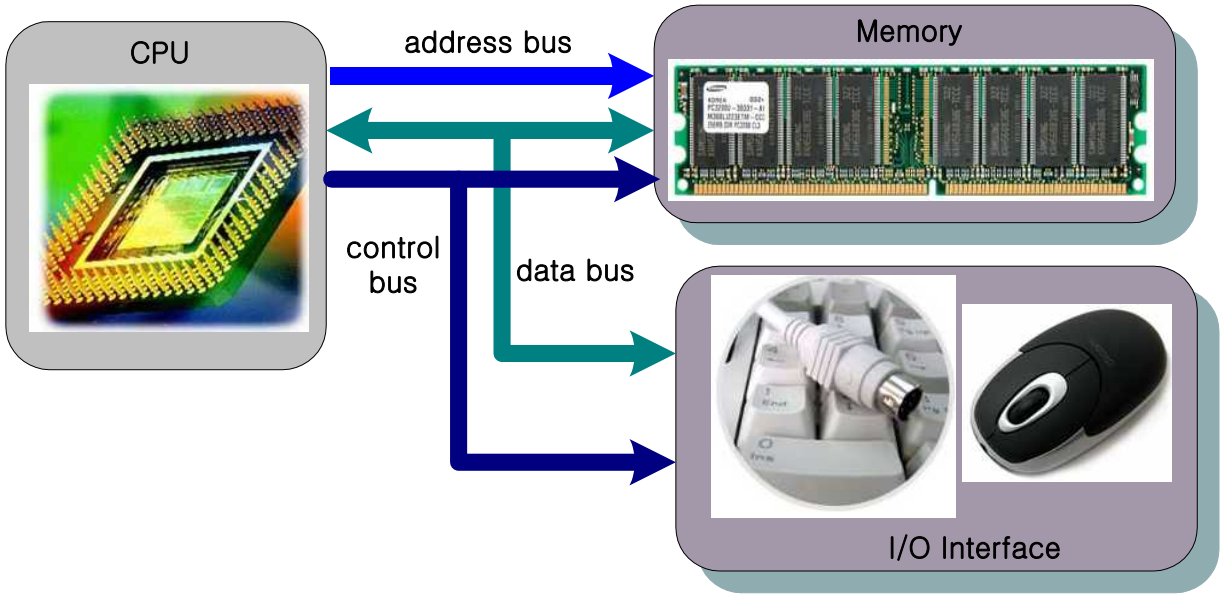


[레지스터 선택 방법 : Clock]

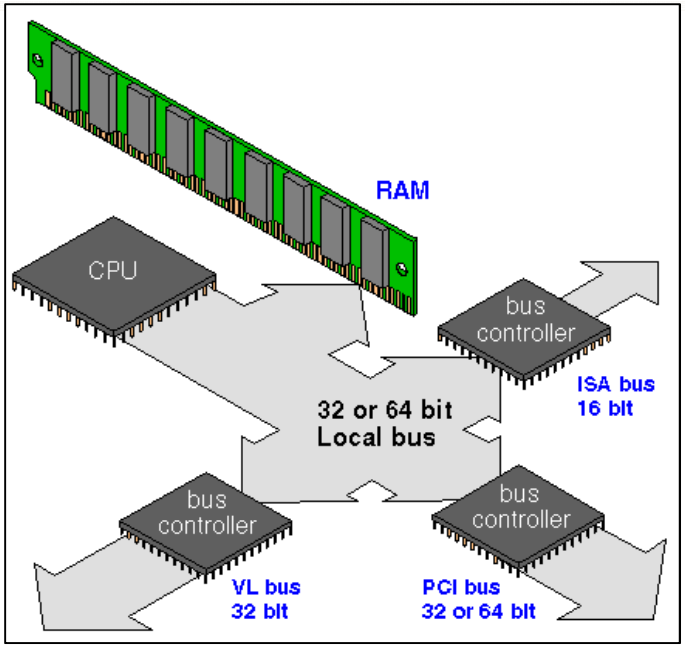
기본 용어-BUS

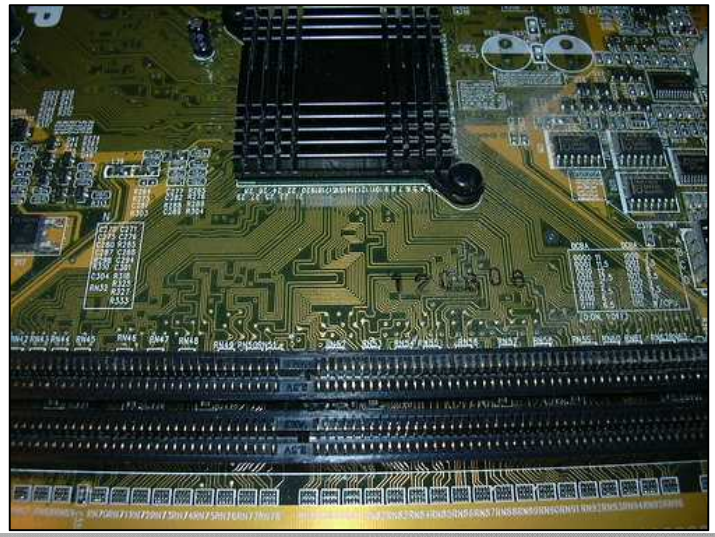
BUS 란?

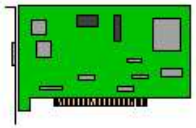
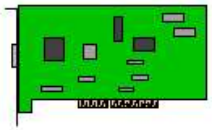
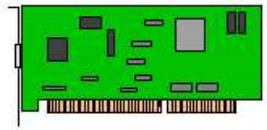
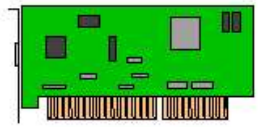
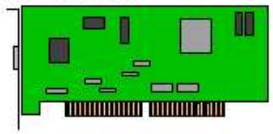
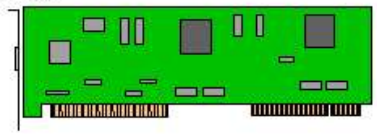
- 정보를 주고 받는 신호선
- 정보의 종류에 따라 구분
- 어드레스 버스, 데이터 버스(양방향), 컨트롤 버스



기본 용어-BUS





PARALLEL PC BUSES	Bandwidth	
	Bits	Speed
PCI 	32 64	33MHz
AGP 	32	66-533 MHz
ISA 	8 16	8-10 MHz
EISA 	32	8-10 MHz
Micro Channel 	32	5-20 MHz
VL-bus 	32	40MHz

마이크로프로세서, 마이크로컨트롤러, 마이크로컴퓨터

마이크로프로세서

- 마이크로(micro)와 프로세서(processor)가 결합된 용어
- '매우 작은'(micro)이라는 의미와 '처리기'(processor)
- 크기가 매우 작고, 뛰어난 계산 능력을 가진 장치
- IC 집적기술, 컴퓨터 구조기술, 시스템 프로그래밍 기술을 함께 묶어 단일 칩으로 집적화한 반도체 소자
- 재료, 수학적 개념, 전자 집약기술, 사회적 요구를 수렴한 다양한 마이크로프로세서가 사용되고 있으며, 앞으로도 꾸준한 연구를 통해 더욱 향상된 마이크로프로세서가 등장
- 프로그램을 신속하게 실행하기 위한 목적으로, 내부 구조를 최적화
- -→ CPU가 가진 기능의 대부분을 하나의 반도체 칩에 집적한 것 MPU(Micro-Processor Unit)라고 부르기도 함

* CPU(중앙처리장치) : 프로그램 명령어를 실행, 제어장치, 연산장치, 레지스터로 구성
* MPU : 컴퓨터의 산술논리연산기, 레지스터, 프로그램 카운터, 명령디코더, 제어회로 등의 연산장치와 제어장치를 1개의 반도체 칩에 모아놓은 처리장치. 주기억장치에 저장되어 있는 명령을 해석하고 실행하는 기능을 한다

마이크로컨트롤러

- 마이크로(micro)와 컨트롤러(controller)가 결합된 용어
- '매우 작은'(micro)이라는 의미와 '제어기'(controller)라는 의미
- **마이크로프로세서의 연산 처리 기능에 제어 기능 추가**
- 프로그램을 실행하면서 장치를 효과적으로 제어하기 위한 목적으로, 내부 구조를 최적화
- 값싼 장난감에서부터 산업용 장치에 이르기까지 넓은 범주를 대상

■ 포함 기능 사례

- ➢ 외부 디지털 전압에 대한 입출력 기능
- ➢ 메모리 기능(플래시 메모리, SRAM, EEPROM 등)
- ➢ 타이머 기능, PWM(Pulse Width Modulation) 펄스 생성 기능
- ➢ 입력신호 캡처 기능, A/D(Analog Digital) 변환 기능
- ➢ 통신 기능 등

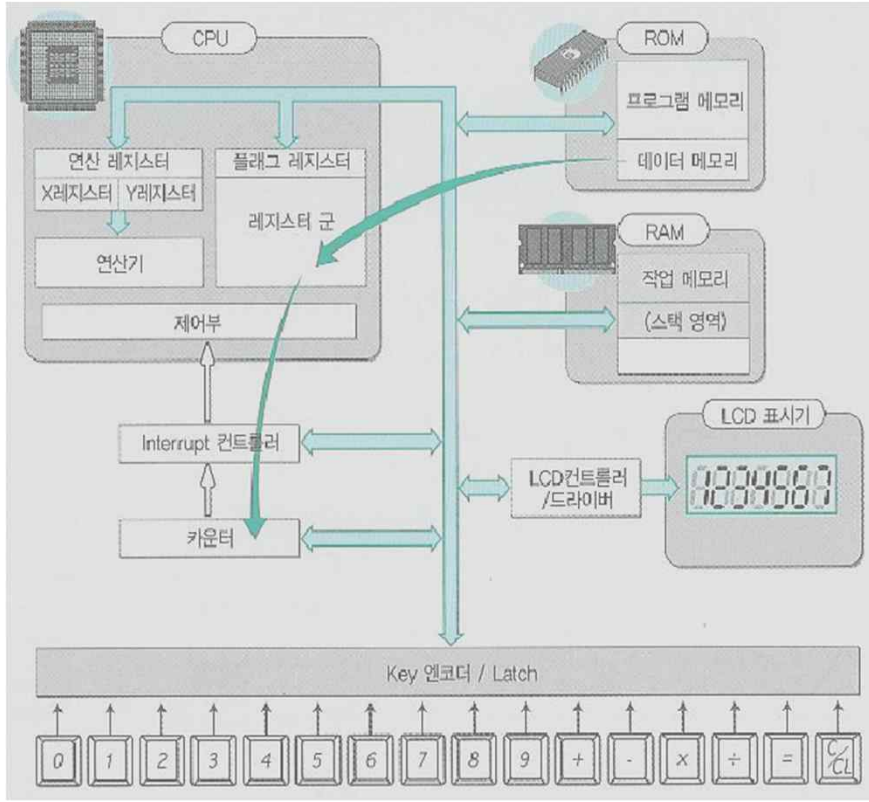
- -→ 마이크로프로세서 중 1개의 칩 내에 CPU 기능 + I/O+ROM+RAM

마이크로컴퓨터

- 마이크로(micro)'와 '컴퓨터(computer)'가 결합된 용어
- '매우 작은(micro)' 컴퓨터 시스템
- 마이크로컴퓨터는 상대적으로 대형 슈퍼컴퓨터와 대비
- (사례) 데스크톱 컴퓨터, 휴대용 노트북, 간단한 제어용 컴퓨터, PMP, 스마트폰 등

컴퓨터의 기본 구성

- 마이크로 컴퓨터의 기본 구성
  - CPU, 메모리, 입력포트, 출력 포트로 구성되며, 서로 데이터 버스라 부르는 신호선에 의해 연결됨.



[마이크로컴퓨터의 기본 구성]

마이크로프로세서의 응용분야

산업	• 모터 제어 • 로봇 제어 • 프로세스 제어 • 수치제어 • 지능형 변환기(transducer)	**계측**	• 의료용 계측기 • 가스 크로마토 그래프 • 오실로스코프
가전제품	• 비디오 레코더 • 레이저 디스크 • 비디오 게임기	**유도제어**	• 미사일 제어 • 지능형 무기 • 우주선 유도 제어
데이터 처리	• 플로터(plotter) • 복사기 • 하드디스크 구동장치 • 테이프 구동장치 • 프린터	**자동차**	• 점화 타이밍 제어 • 연료 분사 제어 • 변속기 제어 • ABS 제어 • 열 방사 제어
통 신	• 모뎀 • 지능형 line card의 제어		

▣ 기본 용어 3

5. ALU (Arithmetic Logic Unit)
6. Accumulator
7. Data Register
8. Address Register
9. PC (Program Counter)
10. IP(Instruction Pointer : Decoder)
11. Controller Sequencer

기본용어 & 마이크로 프로세서의 내부 구조

- ALU (Arithmetic Logic Unit)
  - **산술 또는 논리 연산을 이행하는 곳으로** 두 개의 입력을 가지고 있다. 하나는 accumulator라고 하는 레지스터이고, 또 하나는 데이터 레지스터로 두 개의 입력을 operand라 부른다. 두 개의 operand를 이용하여 덧셈과 뺄셈을 하거나 서로 비교하여 그 결과를 다시 accumulator에 저장한다.
- Accumulator (A, Acc, W)
  - 마이크로프로세서에서 가장 유용한 레지스터이다. 산술과 논리 연산 동안에 가장 기본이 되는 레지스터
- Data Register
  - 데이터 버스로부터 들어오고 나가는 데이터를 일시적으로 저장하는 장소
- Address Register
  - 메모리나 I/O 장치의 어드레스로 나가는 데이터를 일시적으로 저장하는 장소

기본용어 & 마이크로 프로세서의 내부 구조

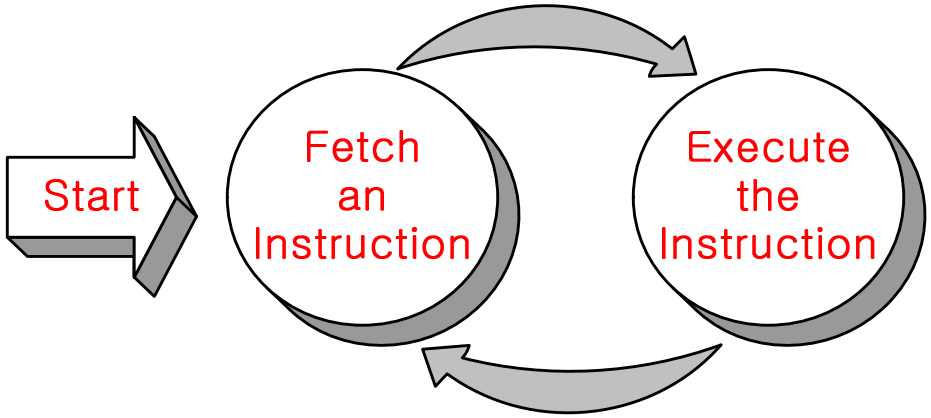
- Program Counter
  - 수행되는 프로그램에 있는 명령의 시퀀스를 컨트롤한다.
  - 즉, 다음 수행해야 될 메모리의 위치를 저장

- Instruction Decoder
  - 명령이 메모리로부터 끄집어내져서 데이터 레지스터에 놓인 후에 그 명령은 이 회로에 의해서 해독된다.
  - 인텔사의 16bit 프로세서 이상에서는 instruction pointer (IP)라고 함.

- Control Sequencer
  - 명령을 수행하기 위해 여러 가지 컨트롤 신호를 만드는데, 각 명령이 다르므로 컨트롤 신호의 다른 조합이 각 명령을 위해 만들어진다. 즉, 명령에 의해 표현된 동작을 하기 위해 필요한 사건의 시퀀스를 결정한다.

마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

- 프로그램 수행 절차의 간단한 예
  - 명령어 수행 사이클
  - **Instruction Cycle = Fetch Cycle + Execute Cycle**
    - **Fetch Cycle : 명령어를 읽어서 해석**
    - **Execute Cycle : 해석한 내용 수행 (데이터 처리하는 것)**



마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

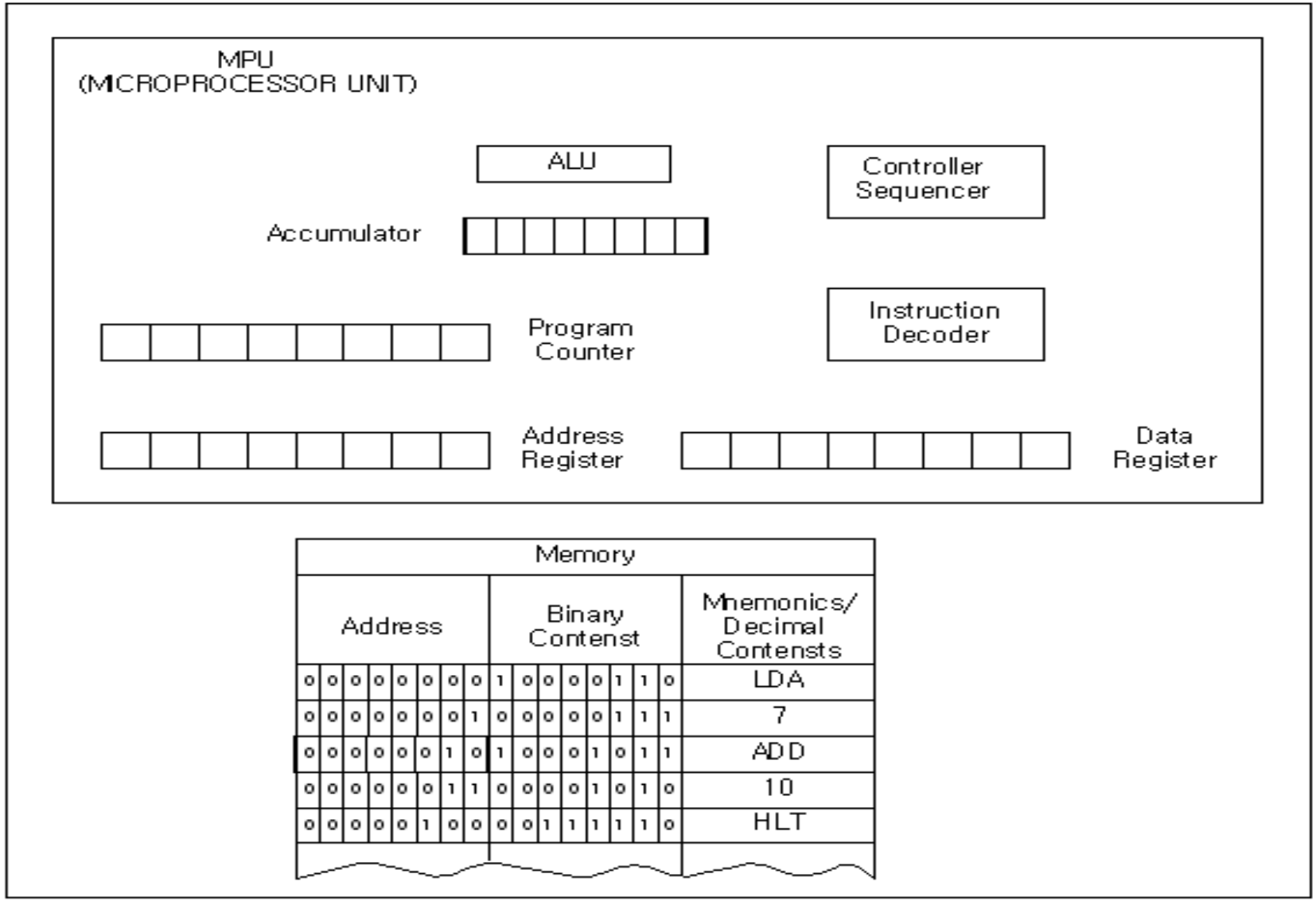
- 프로그램 예

```
LDA   7(10)
ADD   10(10)
HLT
```

- 프로그램의 의미

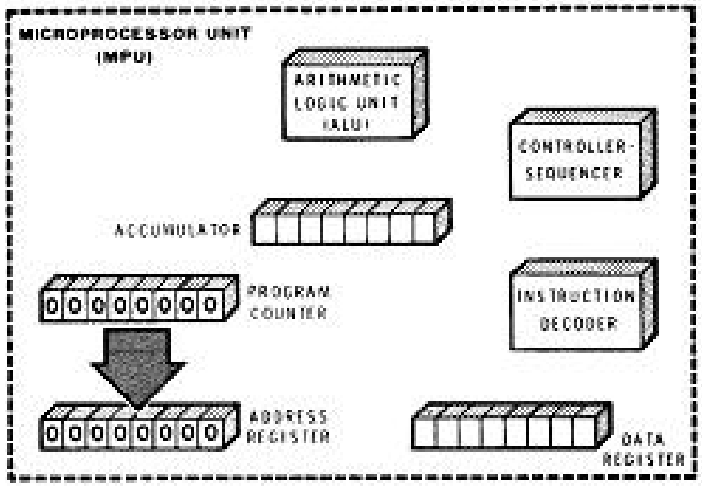
NAME	MNEMONIC	OPCODE	DESCRIPTION
Load Accumulator	LDA	1000 0110(2)	다음 메모리 위치의 내용을 어큐뮬레이터에 로드
Add	ADD	1000 1011(2)	다음 메모리 위치의 내용을 현재 어큐뮬레이터의 내용에 더하고, 그 값을 어큐뮬레이터에 저장
Halt	HLT	0011 1110(2)	모든 동작중지

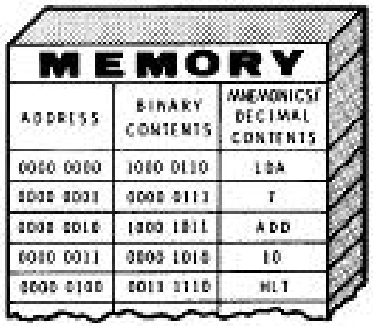
마이크로프로세서의 내부 구조

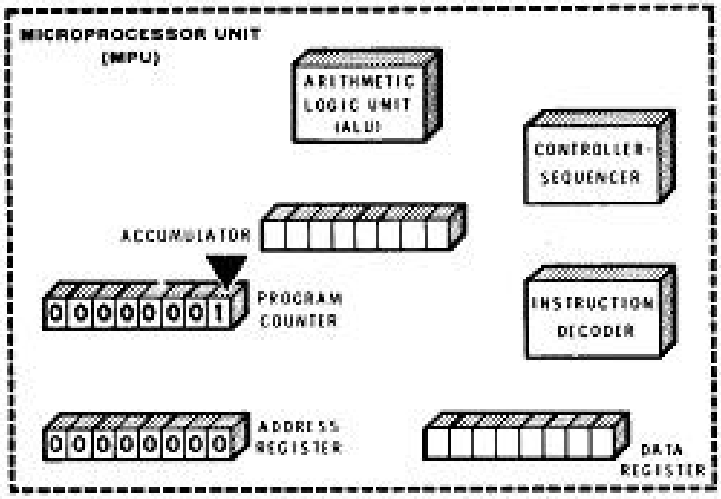


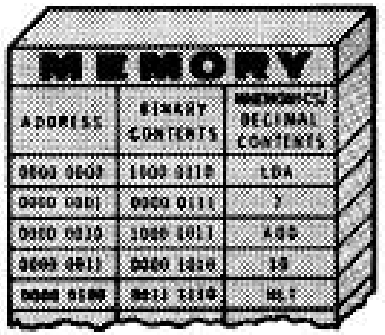
마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

- Fetch Cycle [LDA 7]



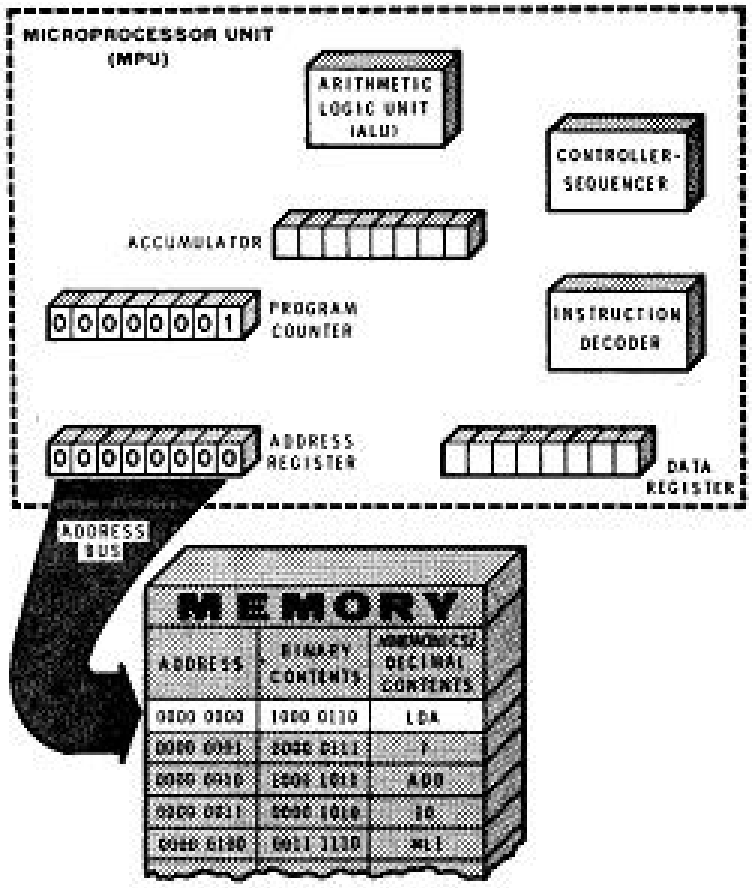


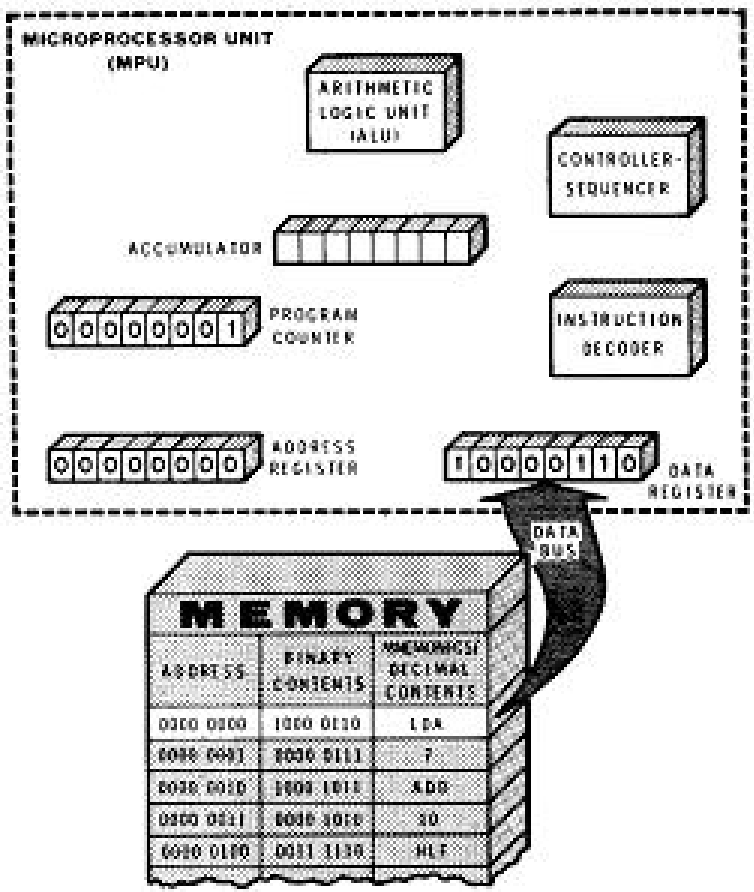




마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

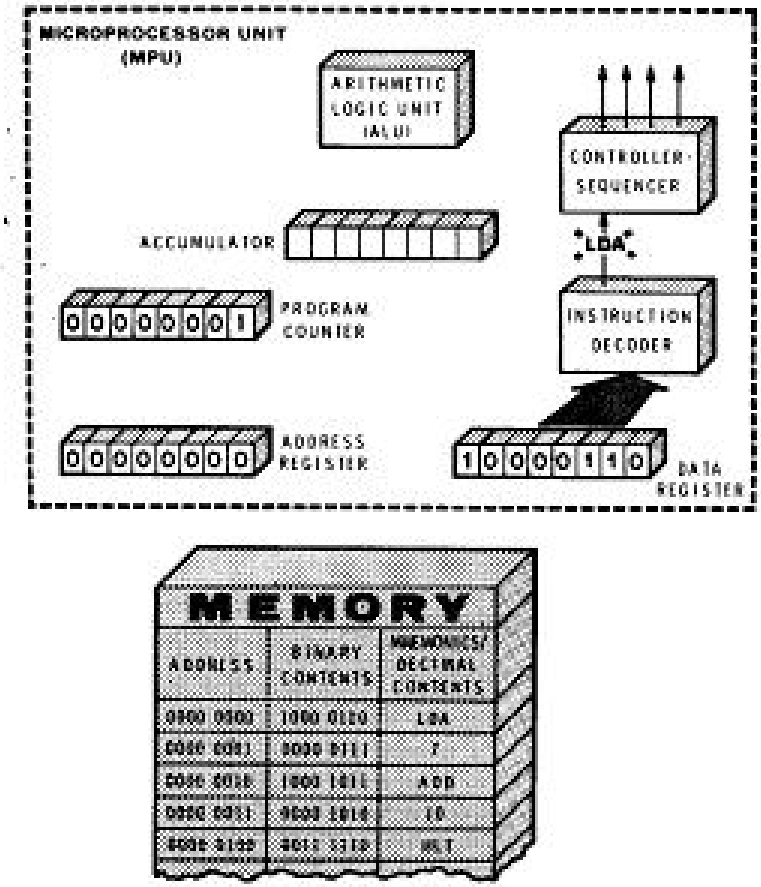
- Fetch Cycle [LDA 7]





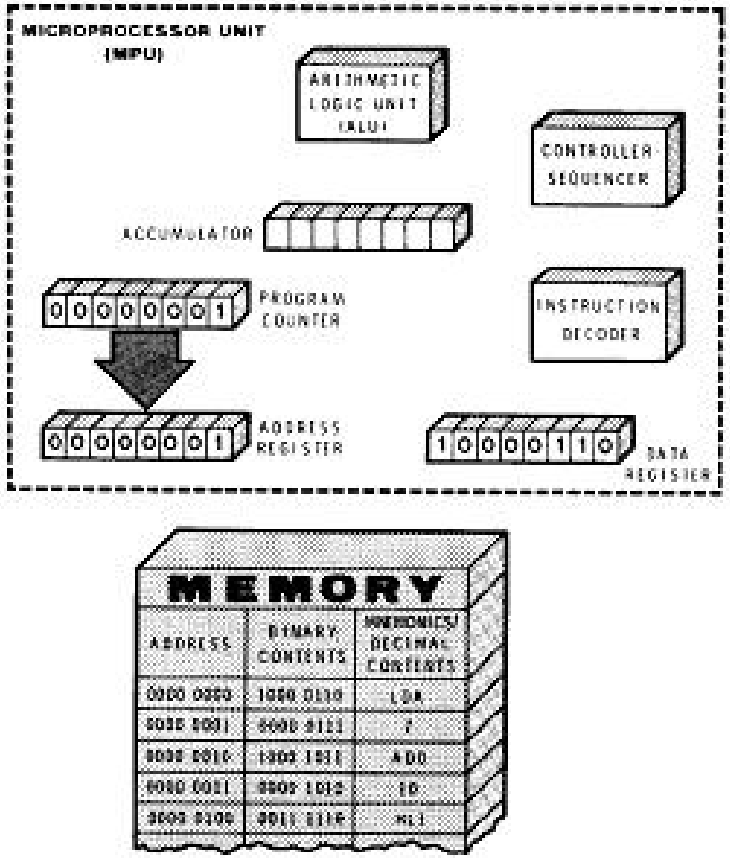
마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

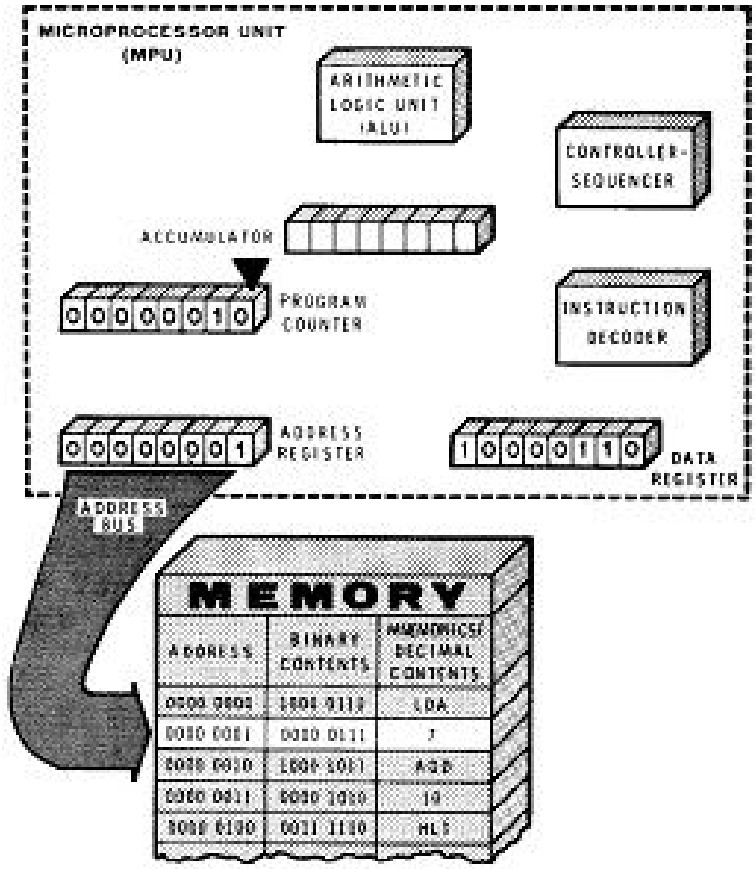
- Fetch Cycle [LDA 7]



마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

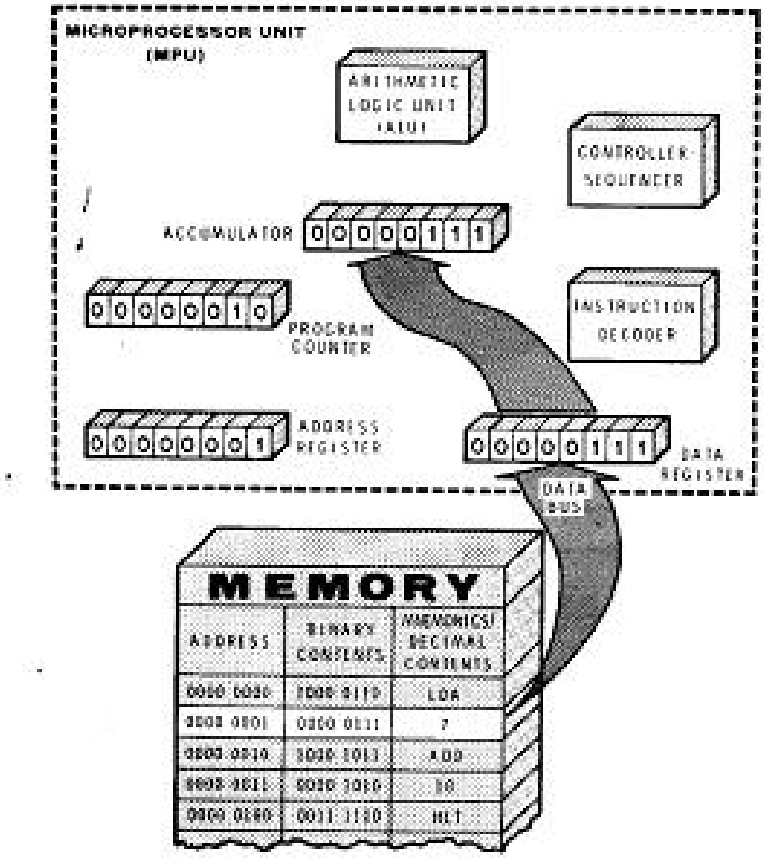
- Execute Cycle [LDA 7]





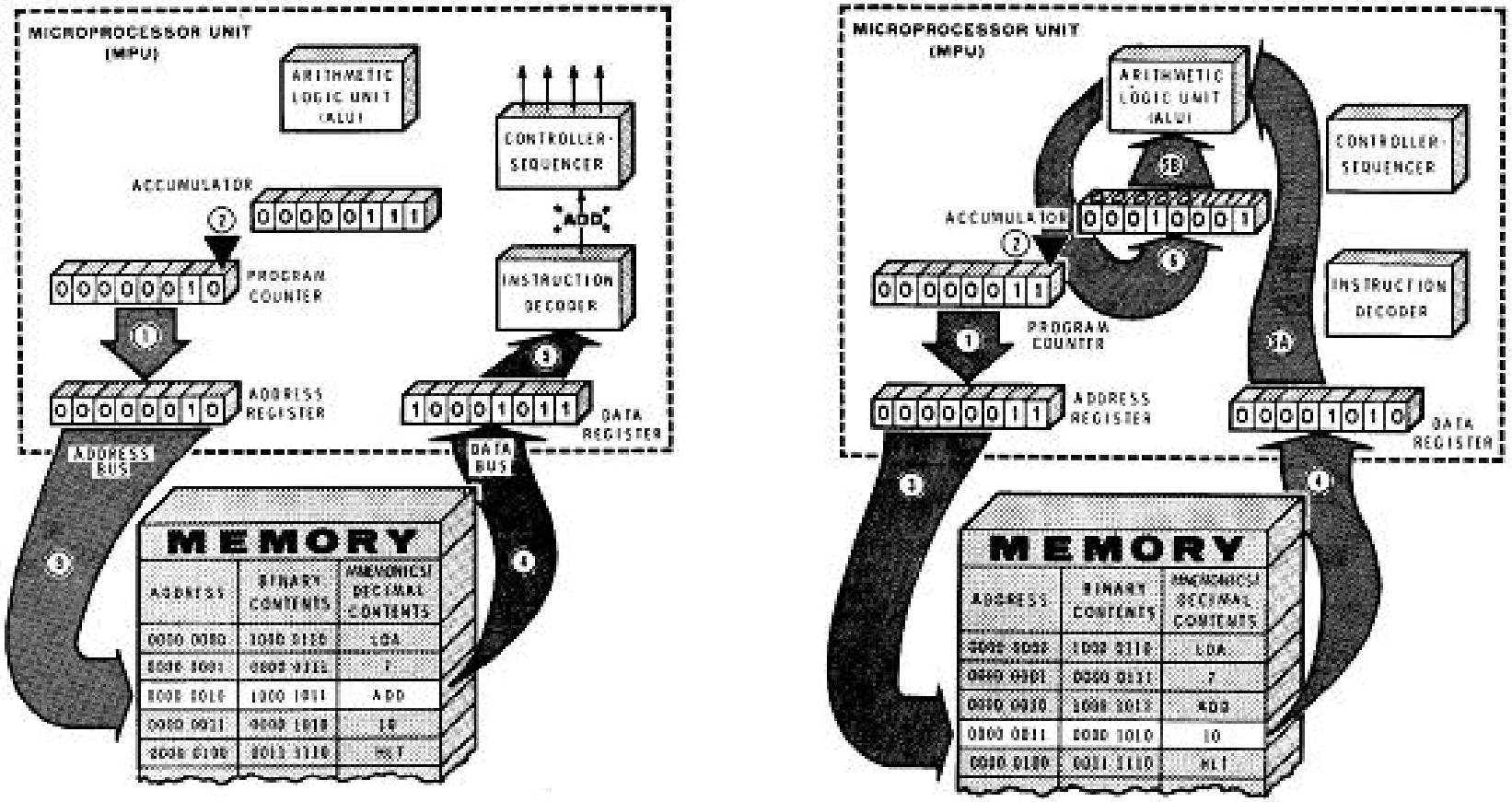
마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

- Execute Cycle [LDA 7]



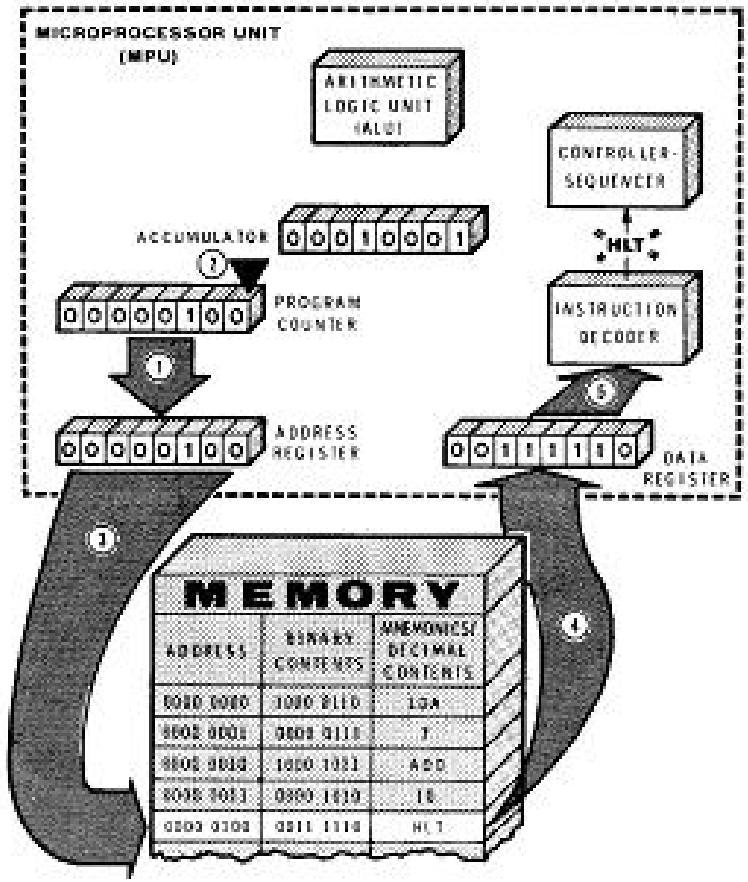
마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

- [ADD 10]



마이크로 프로세서의 내부 구조 – 프로그램 수행 절차

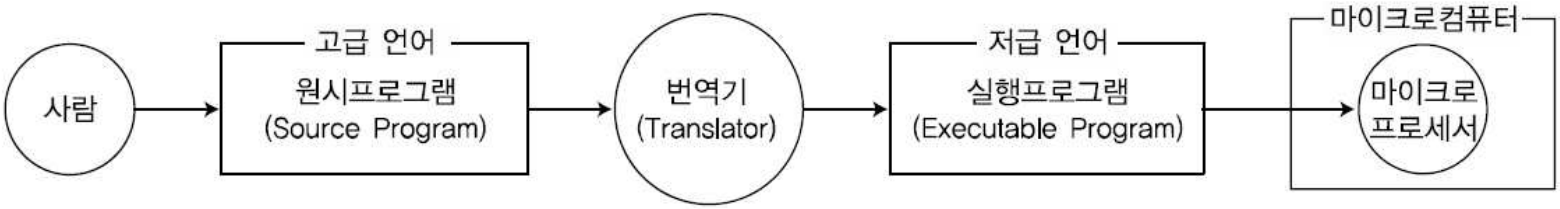
- [HLT]



고급 언어와 저급 언어

■ 번역기의 필요성

- **컴퓨터에 내장된 마이크로프로세서는 2진(0 또는 1)값을 갖는 명령어가 차례대로 실행되면서 작업을 처리**
- **프로그램**
  - 작업을 처리하기 위해 명령어를 차례로 배열한 것



- **원시프로그램 : 사람이 작성한 프로그램(고급 언어로 보통 작성)**
- **번역기(Translator)**
  - 원시프로그램을 저급 언어(Low level language)인 2진 명령어로 변환
  - 컴퓨터가 인식할 수 있는 기계어(Machine language)의 실행프로그램(Executable Program)으로 변환
- **기계어 : 마이크로프로세서 내부의 실행 동작을 명령어 단위로 표현**
  - 명령어를 이해하면 마이크로프로세서 내부 구조를 쉽게 이해

고급 언어와 저급 언어

■ 기계어와 어셈블리어

➢ 기계어(Machine Language)

- 특정 비트에, 특정 의미가 있는 2진값을 설정하는 명령어를 나열

➢ 어셈블리어(Assembly Language)

- 기계어를 사람이 연상하기 쉬운 니모닉(Mnemonic)과 연산 대상이라는 영문 단어로 바꿔 표현

➢ 어셈블러(Assembler)

- 어셈블리어로 작성된 원시프로그램을 기계어로 변환시키는 번역기

어셈블리어	기계어
movw r30, r22	1111 1101 0000 0001
subi r30, 0x00	1110 0000 0101 0000

고급 언어와 저급 언어

■ 원시프로그램에서 실행 파일 생성

➢ **컴파일(Compile), 컴파일러**
- 고급언어로 작성된 원시프로그램을 기계어 목적 파일로 번역하는 과정
- 컴파일러 : 컴파일을 수행하는 번역기

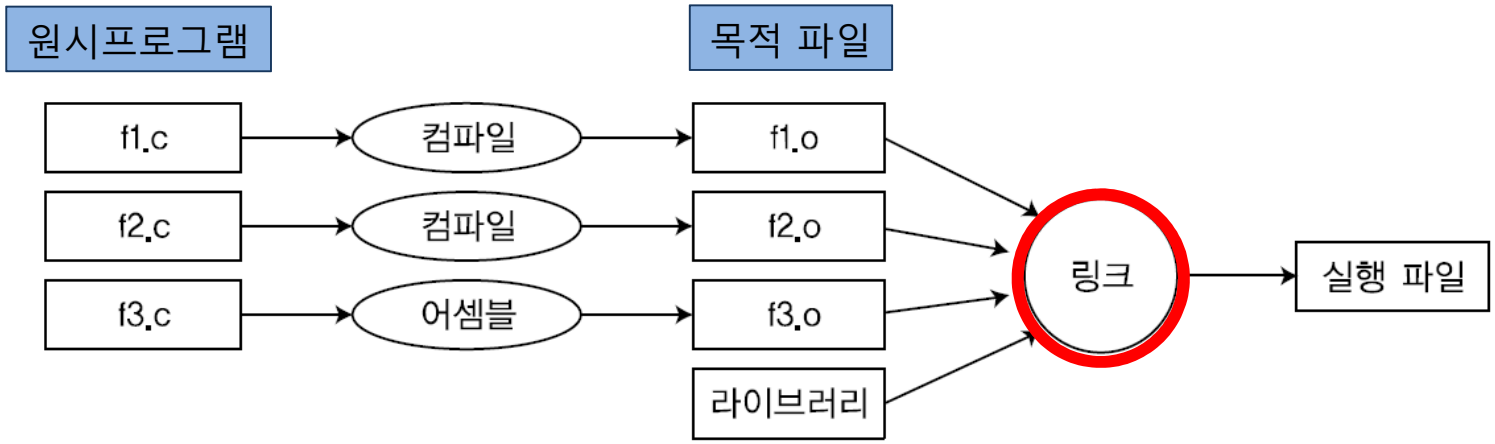
➢ **어셈블(Assemble), 어셈블러**
- 어셈블리어로 작성된 원시프로그램을 기계어 목적 파일로 번역하는 과정
- 어셈블러 : 어셈블을 수행하는 번역기

➢ **링크(Link), 링커**
- 여러 개의 목적 파일을 연결하여 통합된 실행 파일로 만드는 과정
- 링커 : 링크를 전담하는 프로그램

➢ **라이브러리**
- 공통으로 자주 사용하는 기능은 표준화된 이름과 매개 변수가 있는 함수로 작성한 후, 컴파일하여 목적 파일을 생성
- 컴파일된 목적 파일을 모아놓은 파일을 '라이브러리 파일'이라 함
- (예시) sin, cos, tan, printf 등과 같이 자주 사용하는 함수

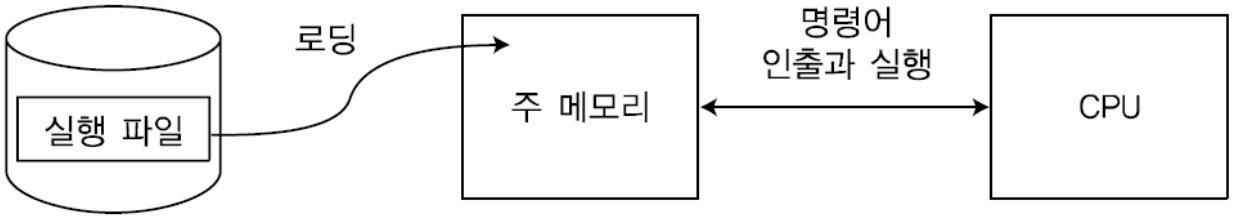
고급 언어와 저급 언어

➢ 실행 파일이 생성되는 과정



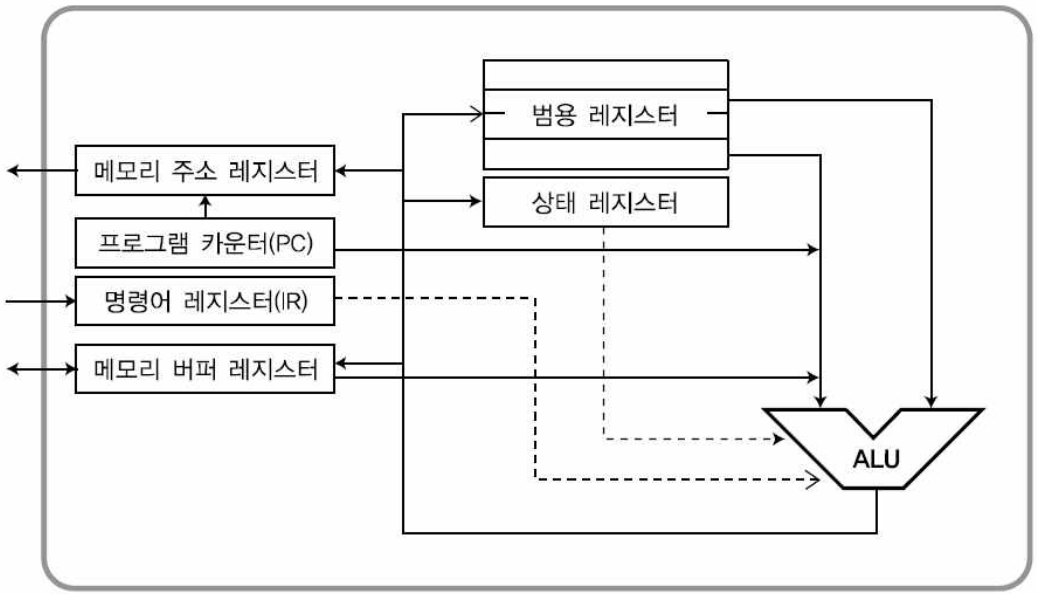
➢ 로딩

- 실행 파일이 중앙 처리 장치(CPU, Central Processing Unit)에서 실행되려면, 주 메모리에 탑재되는 과정 필요
- 로더 : 로딩을 전담하는 프로그램



프로그램 실행 원리

- 특수 레지스터, 범용 레지스터, CPU 외부의 메모리 사이에 ALU(Arithmetic Logic Unit)를 통한 연산을 통해 명령어 실행



특수 레지스터와 범용 레지스터

- **범용 레지스터**
  - CPU 연산을 빠르게 처리하기 위해 ALU와 직접 연결
  - 연산 대상이 되는 오퍼랜드 값을 가짐
- **특수 레지스터**
  - 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
- **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

프로그램 실행 원리

➢ 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

➢ 상태 레지스터(Status Register)

- 명령어를 실행한 후의 연산 결과 정보를 기록
- 기록된 상태 레지스터의 각 비트는 연속되는 다음 명령어 실행에 영향을 미침

1	if(a >= 100)	cpi r24, 0x64
2	a = 0x10;	brcs .+6
3		ldi r24, 0x10

- (예시) brcs .+6 명령은 cpi 명령 실행 후, 상태 레지스터의 캐리 비트 값이 1이면, 현재의 프로그램 카운터보다 6바이트 후의 분기된 곳을 실행

프로그램 실행 원리

➢ **메모리 주소 레지스터(MAR)와 메모리 버퍼 레지스터(MBR)**

- 데이터 인출/기록 과정에서 데이터 또는 주소를 임시로 저장하기 위한 특수 레지스터

■ 명령어 실행

❶ 명령어 인출 단계

- 프로그램 메모리에 있는 명령어를 인출(Fetch)하여, 명령어 레지스터(IR)에 넣는 단계
- 명령어는 연산 동작(opcode), 연산 대상(operand), 연산 결과(result)를 지칭하는 비트 정보로 구성

❷ 명령어 분석 단계

- 명령어 레지스터(IR) 정보를 디코딩하여 레지스터 파일, 버스 라인 등에 필요한 신호를 생성

프로그램 실행 원리

❸ 오퍼랜드 인출 단계

- 디코딩 결과에 따라 연산에 사용될 오퍼랜드(operand) 위치가 결정
- 오퍼랜드가 메모리에 있는 경우, 메모리로부터 인출하여 CPU 내부의 레지스터로 옮기는 단계
- 메모리로부터 CPU 내부로 옮기기 위해서는 부가의 클록이 요구되고 명령어 실행 시간이 길어짐
- 오퍼랜드가 명령어에 포함되어 있거나 레지스터에 있는 경우, 이 단계는 생략

❹ 연산 실행 및 결과 생성 단계

- 명령어 레지스터의 연산 동작(opcode)에 따라 오퍼랜드 값 등을 사용하여 산술 또는 논리연산
- 조합 논리 회로로 구성된 ALU에서 오퍼랜드를 사용하여 연산 결과는 어큐뮤레이터(Accumulator) 또는 범용 레지스터로 기록

❺ 결과 기록 단계

- 명령어를 디코딩한 결과 연산된 결과를 저장하는 위치가 메모리이면, 이 단계가 수행

- 소스코드 작성 방법 (구성요소들)


```
LOOP        MOVF        PORTB, 0        ;PORTB->W
```

Label : 주소를 나타내는 가상의 주소, 실제 주소로서 번역

Mnemonic : 사람들이 쉽게 기억하도록 만들어진 기호(OPCODE)
OP CODE : 그 명령으로 해야 할 일(operation)
Operand : 명령의 대상인 데이터, 번지 또는 레지스터, 없는 경우도 있음

Comment : 주석으로 (;-세미콜론)뒤에 작성

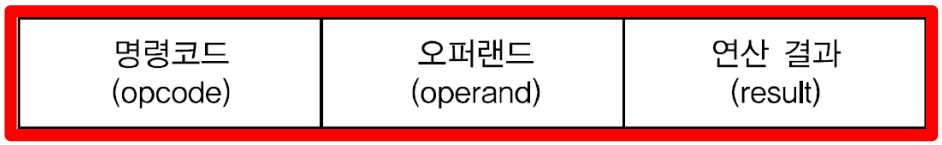
- [Pseudo instruction]어셈블리 프로그램에서 기계어 명령이 아니라 어셈블러에게 어떤 사항을 지시하기 위해 사용되는 명령어. 여기에는 프로그램의 시작과 끝을 나타내는 것, 변수를 위한 기억장소를 확보하는 것, 매크로를 정의하거나 호출하는 것 등이 있다
- [Directive]

프로그램 실행 원리

- **명령어 인출-실행 사이클**
  - 프로그램은 명령어가 메모리에서 인출(Fetch)되어 실행(Execution)되는 사이클이 반복되면서 수행
- **RISC 구조로 실행 단계를 단축할 수 있음**
  - 오퍼랜드 인출 단계를, 별도의 명령어 load
  - 결과 기록 단계를, 별도의 명령어 store
  - 단순해지는 인출-실행 사이클

3주소, 2주소, 1주소, 0주소 마이크로프로세서

- **명령어에 포함되는 비트 필드(bit field) 형식**



  - 명령코드 : 명령어로 실행되어야 할 연산 동작을 표시
  - 오퍼랜드 : 연산에 필요한 데이터 표시
  - 연산 결과 : 명령코드, 오퍼랜드로 얻은 연산 결과 값을 저장할 위치 표시
- **명령어에 포함되는 주소방식에 따라 CPU 내부 구조가 달라짐**
- **3주소 마이크로프로세서 명령어**
  - 명령어에 3개의 오퍼랜드 주소가 포함
  - (예) ADD a, b, c $c \leftarrow a + b$

명령어 셋

➢ 2주소 마이크로프로세서 명령어

- 명령어에 2개의 오퍼랜드 주소가 포함
- (예) ADD a, b `b ← a + b`

➢ 1주소 마이크로프로세서 명령어

- 특수 레지스터 어큐뮤레이터(AC, Accumulator)를 묵시적으로 이용하여 연산 수행
- (예) ADD a `AC ← a + AC`

➢ 0주소 마이크로프로세서 명령어

- 오퍼랜드용 주소를 따로 두지 않음
- 스택(stack)에 있는 연산 대상의 데이터를 사용
- (예) ADD

명령 순서	연산 동작
PUSH #a	SP ← SP + 1, [SP] ← #a
PUSH #b	SP ← SP + 1, [SP] ← #b
ADD	AC ← [SP], SP ← SP − 1 AC ← AC + [SP] [SP] ← AC

명령어 셋

■ 어드레싱 모드

- **연산을 위한 오퍼랜드를 지칭하는 방법**
- **즉시 주소방식**
  - 명령어에 연산 대상이 되는 데이터 값을 갖고 있음

명령어 예시	연산 동작
ADD #23	AC ← AC + 23

- **직접 주소방식**
  - 명령어에 포함된 주소는 주 메모리의 연산 대상 데이터에 대한 주소

명령어 예시	연산 동작
ADD a	AC ← AC + [a]

- **간접 주소방식**
  - 명령어에 기록된 주소는 오퍼랜드를 인출할 수 있는 주소

명령어 예시	연산 동작
ADD [a]	MAR ← [a], AC ← AC + [MAR]

명령어 셋

➢ 레지스터 직접 주소방식

- 명령어에 포함된 주소는 레지스터 파일 중 하나를 지칭하는 주소

명령어 예시	연산 동작
ADD R1	AC ← AC + R1

➢ 레지스터 간접 주소방식

- 명령어에 포함된 주소는 레지스터 파일의 레지스터 지칭
- 레지스터 값은 주 메모리의 주소, 연산 대상 오퍼랜드는 이 주소로부터 인출

명령어 예시	연산 동작
ADD [R1]	MAR ← R1, AC ← AC + [MAR]

명령어 셋

➢ 변위 주소방식

- 명령어에 두 개의 주소 필드 포함
- 하나는 베이스(또는 인덱스) 레지스터, 다른 하나는 변위(displacement)
- 배열 또는 메모리의 특정 위치를 베이스(또는 인덱스) 레지스터를 기준으로, 변위 차를 갖는 메모리의 오퍼랜드 주소로 계산

명령어 예시	연산 동작
ADD R1, disp	MAR ← R1 + disp, AC ← AC + [MAR]

➢ 상대 주소방식

- 변위 주소방식과 유사하지만, 베이스 레지스터로 프로그램 카운터 사용

명령어 예시	연산 동작
BNE disp	if Not_Zero then PC ← PC + disp

명령어 셋

■ 명령코드 동작

- **명령코드 필드의 주요 동작은 연산, 데이터 이동, 프로그램 흐름제어**
- **연산**
  - 덧셈, 뺄셈, 곱셈, 나눗셈과 같은 산술연산
  - OR, AND, XOR, 좌로 비트천이, 우로 비트천이, 보수 변환 등의 논리연산
- **데이터 이동**
  - 레지스터-레지스터, 레지스터-메모리, 메모리-메모리 데이터 이동 명령
- **프로그램 흐름제어**
  - 프로그램의 흐름을 바꾸는 분기 명령으로, 프로그램 카운터(PC) 값이 변경되면, 명령어 인출 위치가 바뀌고 프로그램 흐름이 달라짐

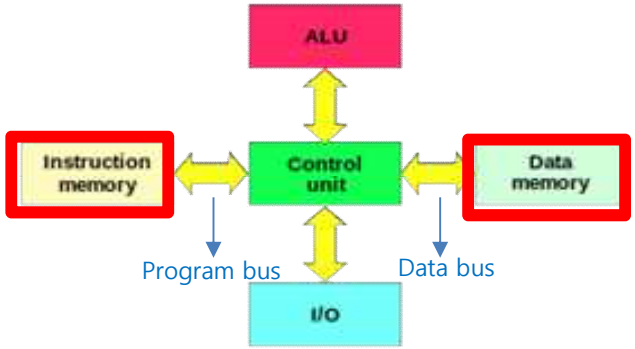
저장형 프로그램 컴퓨터

■ 저장형 프로그램 컴퓨터

- 폰 노이만 아키텍처(von Neumann architecture)로 설계된 컴퓨터 모델을 '저장형 프로그램(Stored Program) 컴퓨터'라 함
- 메모리에 명령어와 데이터를 저장하고 읽기와 쓰기 동작을 반복하면서 원하는 연산을 수행
- 제어 신호는 명령어 인출-실행 사이클(Fetch-execution cycle) 과정에 필요한 데이터 흐름과 연산 제어

Harvard architecture & Von Neumann architecture

Harvard architecture



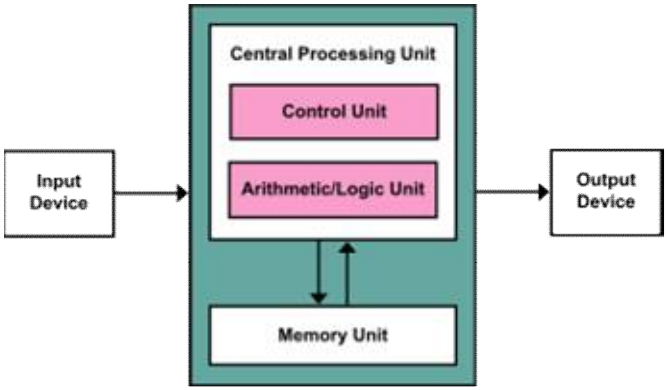
The Harvard architecture is a computer architecture with physically separate storage and signal pathways for instructions and data.

Under pure von Neumann architecture the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system. **In a computer using the Harvard architecture, the CPU can both read an instruction and perform a data memory access at the same time**, even without a cache. A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

Also, a Harvard architecture machine has distinct code and data address spaces: instruction address zero is not the same as data address zero. Instruction address zero might identify a twenty-four bit value, while data address zero might indicate an eight-bit byte that is not part of that twenty-four bit value.

명령어길이(14bit)와 데이타길이(8bit)다름

Von Neumann



This describes a design architecture for an electronic digital computer with parts consisting of a processing unit containing an arithmetic logic unit and processor registers, a control unit containing an instruction register and program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms.[1][2] The meaning has evolved to be any stored-program computer in which **an instruction fetch and a data operation cannot occur at the same time because they share a common bus.** This is referred to as the von Neumann bottleneck and often limits the performance of the system.[3]

저장형 프로그램 컴퓨터

■ 저장형 프로그램 컴퓨터의 한계

➢ 반도체와 컴퓨터 기술 발전

- 메모리 용량 증가
- CPU 속도도 향상

➢ CPU와 메모리 사이의 병목 현상(Bottle-neck)

- CPU 구조 기술이 발달할수록, 처리되는 데이터가 많아질수록,
  메모리와 CPU 사이에 더 많은 데이터 왕래가 필요함
- 단일 CPU의 저장형 프로그램 컴퓨터는 정해진 연결선만을 통해,
  명령어와 데이터를 메모리로부터 전송
- CPU와 메모리 사이의 한정된 데이터 전송 속도
- CPU 연산 능력이 향상되어도, 병목 현상으로 향상된 연산 능력에 대한 제약
- 한계를 연장하기 위해 메모리 계층을 이용하거나 병렬 처리 컴퓨팅

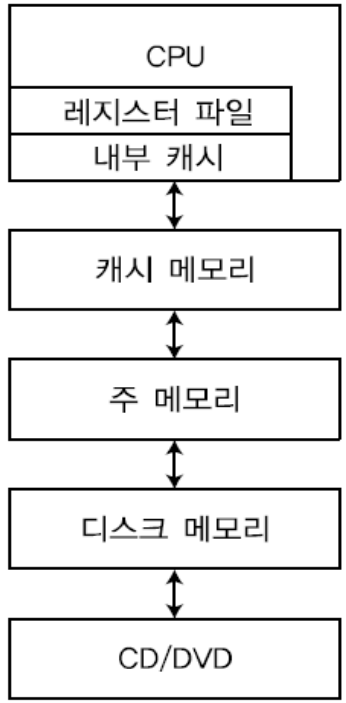
저장형 프로그램 컴퓨터

■ 메모리 계층의 주요한 2가지 기능

➢ **메모리의 병목 현상과 메모리 속도, 속도 대비 가격을 고려하여 메모리를 계층적으로 배치**

➢ **지역성 원리(Principle of locality)에 따라 곧 처리될 프로그램과 데이터 일부를 CPU 근처의 속도가 빠른 메모리에 놓고 처리**

- 지역성(locality)에 의해 참조되는 데이터 영역은 특정 부위에 확률적으로 집중
- 자주 참조되는 데이터를 빠른 캐시에 놓고 사용



[메모리 계층(Memory Hierarchy)]

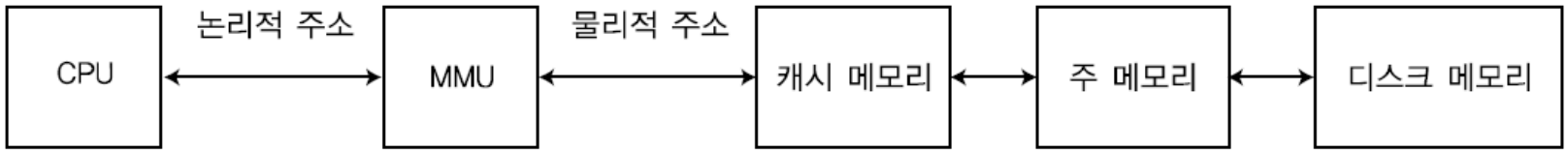
저장형 프로그램 컴퓨터

■ 가상 메모리 (Virtual Memory)

- **실행될 프로그램 또는 데이터는 디스크 메모리에 두었다가 실행이 시작되면 주 메모리(Main memory)로 이동**
  - 모든 프로그램과 데이터를 이동시키지 않고, 사용되는 일부만을 이동하여 사용
  - 프로그램 또는 데이터가 더 요구되는 상황이 발생하면 추가로 이동하여 사용
- **가상 메모리로 확장된 메모리 공간은 운영체제(Operation System)가 관리**

저장형 프로그램 컴퓨터

- **논리 주소를 물리 주소로(하드웨어적으로) 빠르게 변환하여 사용**
  - CPU는 논리 주소로 프로그램 또는 데이터에 접근
  - MMU에서는 논리 주소를 주 메모리에 적재된 물리 주소로 TLB(Translation Lookaside Block)라는 테이블을 이용하여 빠르게 변환
  - TLB에서 요구하는 데이터가 주 메모리에 적재되어 있지 않으면, 페이지 펄트(Page Fault)가 발생하고,
  - 디스크 메모리에서 주 메모리로 옮겨짐(운영체제의 페이지 단위)

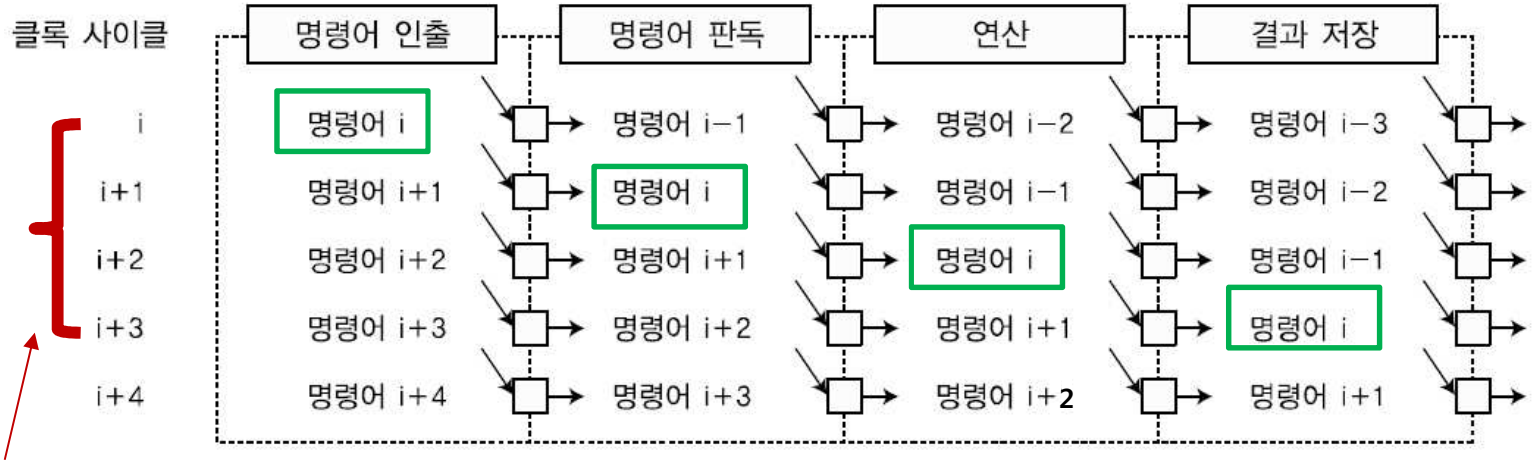


[MMU를 이용한 메모리 계층]

RISC와 CISC

■ RISC

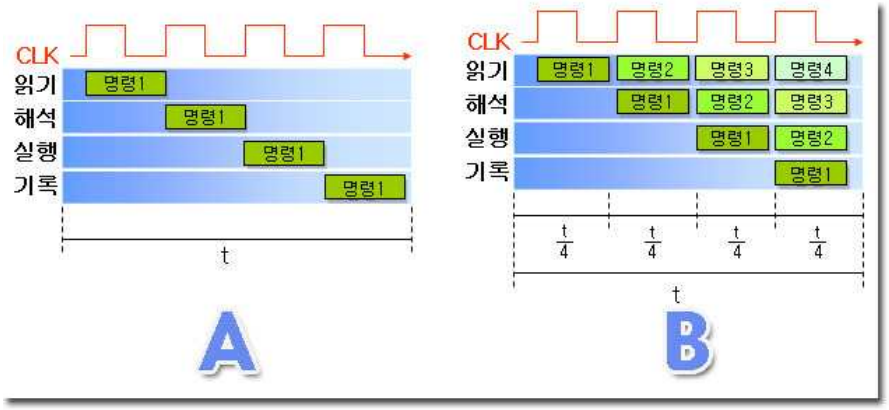
- RISC(Reduced Instruction Set Computer)는 기존의 컴퓨터 구조를 CISC(Complex Instruction Set Computer)로 규정하고, 이와 상대되는 개념
- 명령어 셋을 줄임으로써 CPU 하드웨어 구조를 CISC와 비교하여 간단히 만들 수 있다는 취지에서 시작
- 파이프라인 기법(Pipelining)을 이용하여, 한 사이클에 한 명령어 실행



- 최상의 경우 평균 한 사이클마다 한 개의 명령어가 실행 (명령어 I 실행완료)
  - i+2 클록 스냅샷 : 명령어 i+2는 명령어 인출 단계를, 명령어 i+1은 명령 판독 단계를, 명령어 i는 연산 단계를,명령어 i-1은 결과 저장 단계를 수행

➢ 파이프라인 기법(Pipelining)

: 데이터 처리단계의 출력이 다음 단계의 입력으로 이어지는 형태로 연결된 구조를 가리킨다. 이렇게 연결된 데이터 처리 단계는 여러 단계가 서로 동시에, 또는 병렬적으로 수행될 수 있어 효율성의 향상을 꾀할 수 있다.



_CLK(클럭)은 시간의 흐름이라 생각하면 된다(좌 -> 우). 한 명령은 4단계를 거쳐 진행된다고 가정한다(상 -> 하).

(B가 파이프라인 구조)

A의 처리구조에선 1 t의 시간동안 1개의 명령을 입력받아고 1개의 명령을 처리했다.

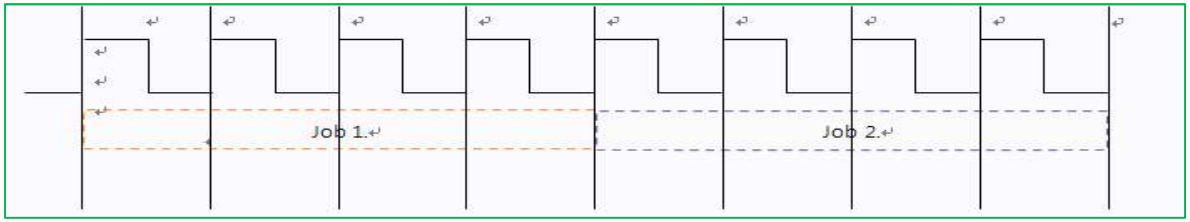
B의 처리구조에선 1 t의 시간동안 4개의 명령을 입력받았고 1개의 명령을 처리, 그리고 3개의 명령이 처리되는 중이다.

그림의 경우는 각 처리단계에 소요되는 시간이 동일하고 처리단계가 전이되는데 발생하는 비용이 없는 이상적인 파이프라인을 예시로 들었다. 프로세서가 작업하는 시간을 최대한 사용하여 작업의 능률을 높이는 기법이다.

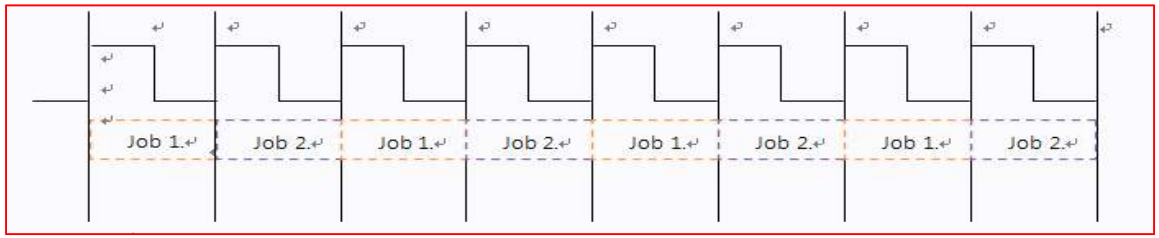
RISC와 CISC 비교

➢ 파이프라인 기법(Pipelining)

첫번째 그림은 파이프라인이 적용되지 않은 그림. 두 개의 작업이 CPU 에 입력된 순서대로 이루어짐. 이렇게 되면 Job1 과 Job2 가 완료되는 시간이 많이 차이 나게 됨.



두 번째 그림은 파이프라인을 적용한 그림. 파이프라인의 적용 유무와 관계없이 작업이 완료되는 시간은 똑같지만 Job1 이 끝나고 1cycle 시간 후에 바로 Job2 가 끝나므로 수많은 작업을 처리해야 하는 큰 프로그램의 경우 바로바로 작업이 완료되게 되어 효율성이 커지게 된다.



-→ 자동차 생산라인이나 세탁소에 많이 비유. 자동차가 생산되기 시작되는 시간은 다소 걸리지만 한 번 생산이 되기 시작하면 짧은 주기로 계속해서 생산이 됨. 또한 세탁소로 비유하자면 CISC 세탁소는 하나의 기계로 세탁과 탈수를 모두 하기 때문에 시간이 오래 걸리지만 RISC 세탁소를 세탁과 탈수 기계가 따로따로 있어서 세탁시간이 줄어들게 됨.

한 가지 유의할 점은 입력되는 데이터의 크기가 일정해야 파이프라인의 효과가 극대화된다는 점입니다. CISC 프로세서의 명령어들은 길이가 다양하고 복잡하여 이러한 파이프라인을 적용할 수 없다. RISC 프로세서의 명령어들은 단순하고 길이가 일정해서 파이프라인을 적용하는 데 있어 최적이다.

RISC와 CISC

- **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
- **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

➢ 데이터 종속 보장을 위한 nop 명령 삽입 기능

```
load r1, 100
add r0, r1
```

- 파이프라인 구조에서 load 명령의 결과 저장 단계가 끝나야 r1에 주소 100의 오퍼랜드 값이 이동
- 연속된 add 명령어의 연산 단계에서는 직전 명령의 결과가 데이터로 갱신되지 않았기 때문에 연산을 계속할 수 없음
- 컴파일러는 문제 해결을 위해 nop 명령을 삽입하여, r1에 값이 갱신될 때까지 시간 지연

```
load r1, 100
nop
add r0, r1
```

➢ 선인출(Pre-fetch)과 예측 실행

- branch 명령은 파이프라인의 최종 단계까지 실행되어야만, 다음 명령어의 위치를 확인할 수 있음
- 다음에 실행될 명령어 위치가 늦게 결정되어도, 연속된 명령어가 결정될 것으로 예측하여 선인출하여 실행
- 연속된 명령어로 분기 위치가 결정되면, 수행된 파이프라인 단계만큼 시간을 절약한 효과가 있음

➢ RISC형 마이크로프로세서에 적합한 컴파일러

- RISC형 마이크로프로세서는 짧은 명령어 길이, 파이프라인 구조, 간단한 어드레싱 모드, 데이터 종속 보장을 위한 nop 명령어 삽입 등의 특징을 고려한 컴파일러 설계가 필요함

RISC와 CISC

■ CISC

➢ **CISC는 RISC와 상대되는 단어로, RISC 개념이 만들어지면서 CISC 용어가 생성**

- **가변 길이의 명령어 형식**
- 명령어의 길이가 길다
- **다양한 어드레싱 모드를 사용**
- 복잡한 연산 동작을 수행할 수 있는 다양한 명령어로 구성

➢ **현대에는 마이크로프로세서가 고성능화되면서, RISC와 CISC의 구별이 점점 모호해지고 있음**

Thank you!