Computer Architecture

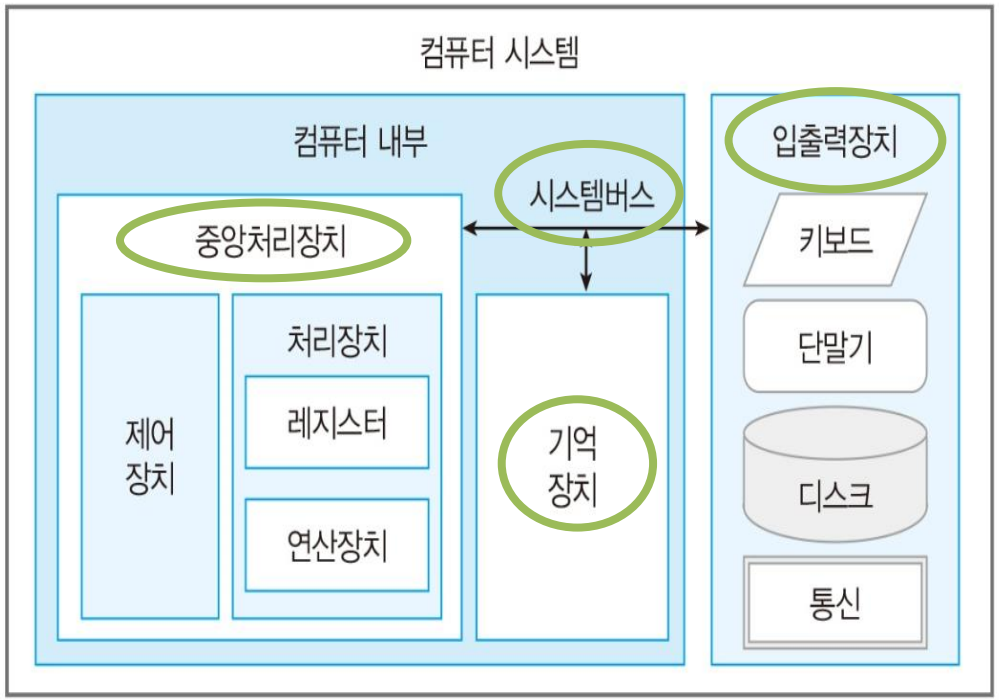
02 CHAPTER

중앙처리장치

1. 컴퓨터시스템 구성

- **컴퓨터 구조 : 프로그램 내장형 구조(폰 노이만 구조)**
  - **프로그램과 데이터를 주기억장치 안에 저장하는 것**

- **컴퓨터시스템 구성 요소**



- **외부 세계와의 데이터 전송을 담당한다.**
- **구성 요소들을 연결한다.**
- **프로그램을 실행하며 컴퓨터의 모든 동작을 제어한다.**
- **프로그램과 데이터를 저장한다.**

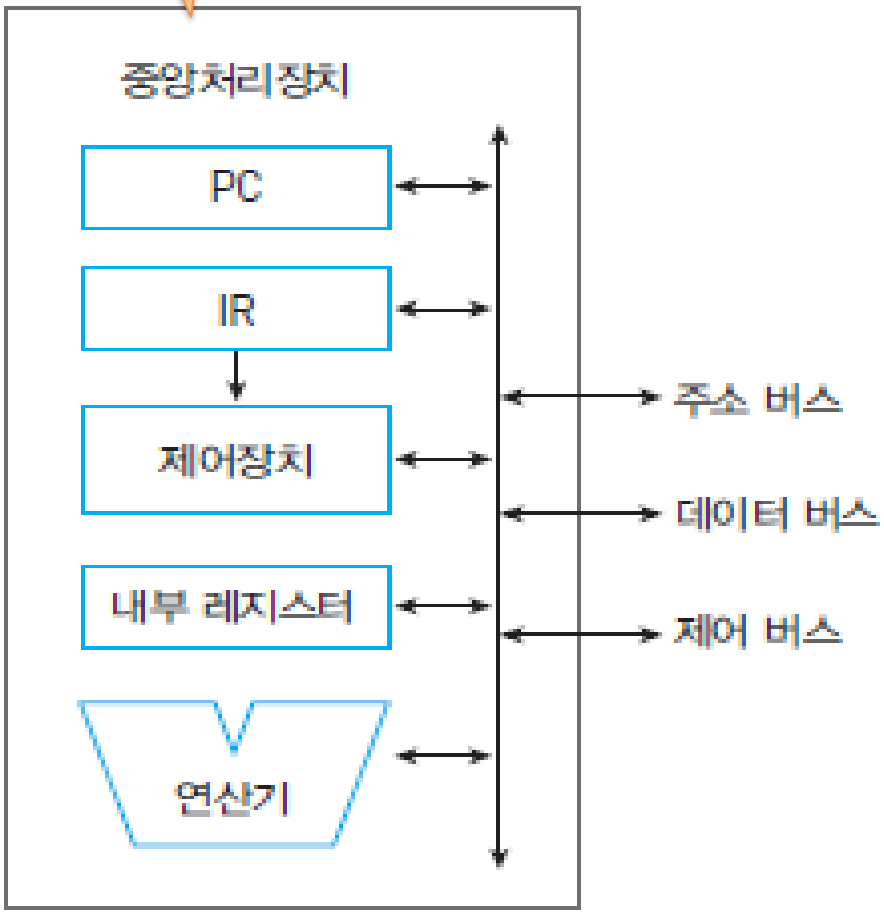
- **중앙처리장치**
  - 기억장치에 저장된 명령어들을 하나씩 가져와 명령어의 의미를 해석하고 데이터를 처리
- **기억장치**
  - 프로그램과 데이터를 저장하는 기능
- **입출력장치**
  - 사용자 혹은 컴퓨터 외부에서 프로그램과 데이터를 컴퓨터에게 공급하고 처리 결과를 받아가는 기능
- **시스템 버스(system bus)**
  - 컴퓨터의 구성 요소인 중앙처리장치, 기억장치, 그리고 입출력장치를 연결하는 여러 개의 선으로 구성된 데이터 전달 경로

2. 컴퓨터의 구성 요소

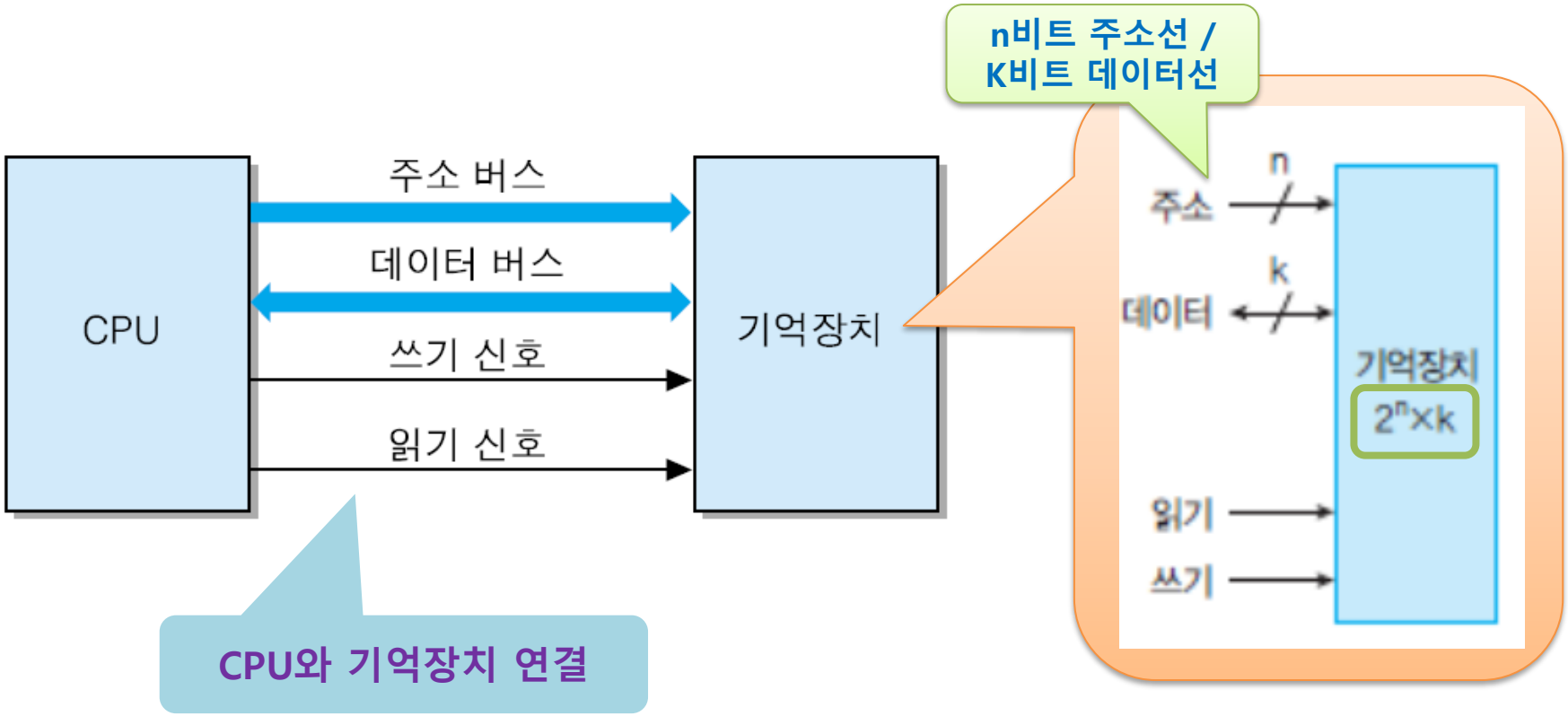
- **중앙처리장치**
  - CPU(Central Processing Unit), 마이크로프로세서
  - 수 십억개의 반도체회로로 구성
  - 프로그램을 실행하는 역할

- **명령어사이클(단순화한구조. 추가설명 8.명령어사이클)**
  - 인출단계 : 주기억장치에서 명령어를 가져오는(인출) 단계
    - 가져올 명령어의 주소와 가져온 명령어를 저장할 장소 필요
    - 프로그램카운터(PC)와 명령어레지스터(IR)
  - 실행단계 : 명령어를 해독하여 명령어를 실행하는 단계
    - 명령어를 해석하고, 이에 맞게 연산을 실행하며 필요한 데이터를 저장할 장소 필요
    - 제어장치(CU), 연산기(ALU), 내부레지스터

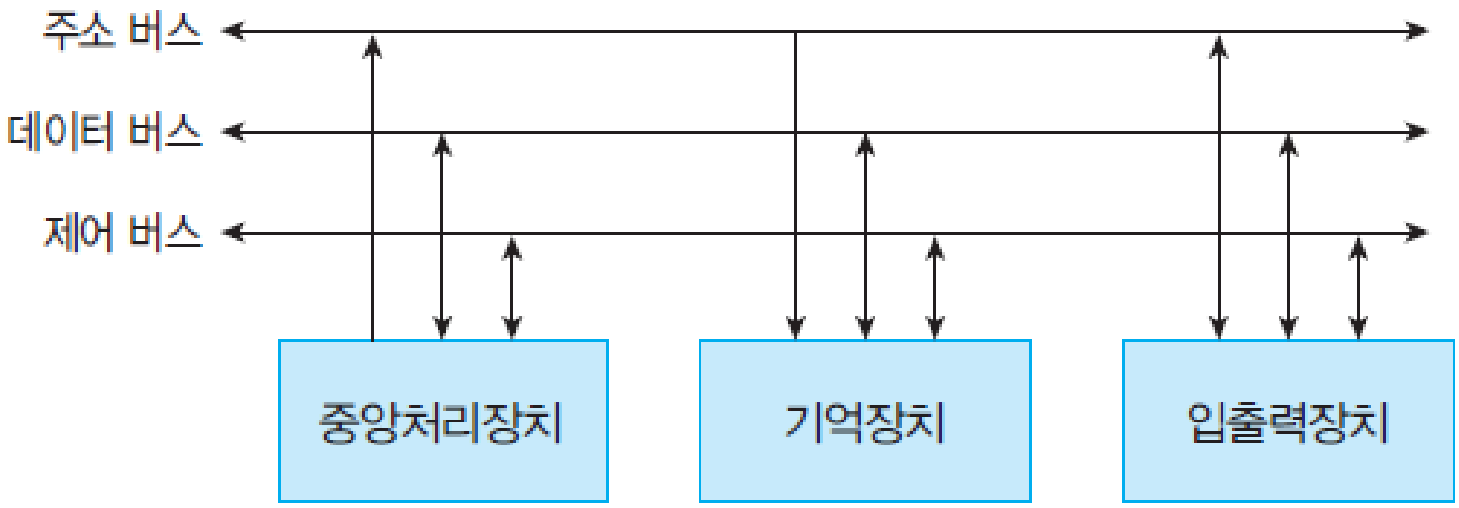
중앙처리장치의 구조



- **주기억장치**
  - **Main Memory**
  - **중앙처리장치가 처리해야 할 프로그램과 데이터 저장**
  - **주소에 의하여 내부 데이터를 액세스하는 거대한 1차원 배열**



- **시스템버스**
  - 컴퓨터의 구성 요소들을 연결하는 신호선들의 모임
  - CPU, 메모리, I/O 장치 등과 상호 필요한 정보를 교환하기 위해 연결하는 공동의 전송선
  - 한개의 신호선, 한비트 정보전달(즉 16비트전달,16개전송선 필요)



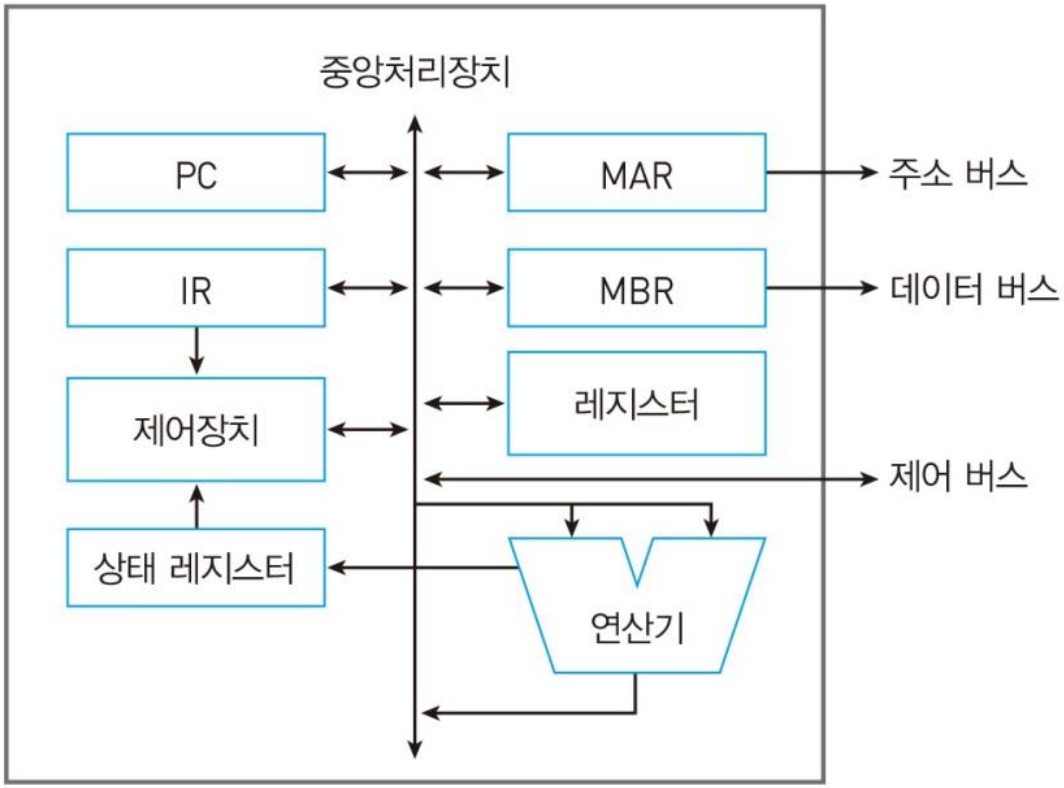
- 주소선(=번지버스)
  - 기억장치에 대한 입력 신호로 작용
  - 기억장치 안에 포함된 여러 개의 기억장소 중 하나를 지정
  - N비트의 주소선 → 2^N개의 기억장소(=워드) 포함

- 데이터선(=자료버스)
  - 데이터를 주고받는 용도
  - K비트의 데이터선 → 한 개의 기억장소에 K비트 저장
  - 데이터선비트수 = 컴퓨터의 워드word크기

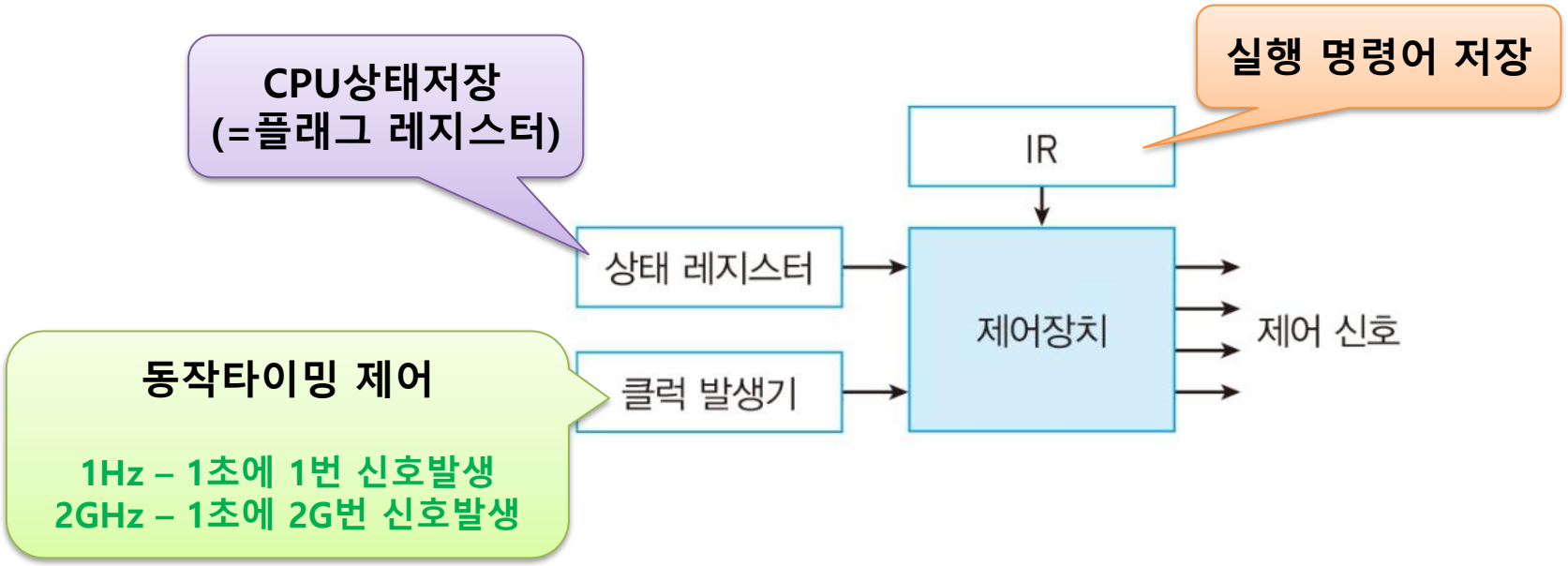
- 제어선(=제어버스)- 읽기, 쓰기
  - 읽기신호 : 주소선으로 지정된 기억장소에 저장된 데이터가 데이터선으로 읽혀 나온다.
  - 쓰기신호 : 주소선으로 지정된 기억장소에 데이터선으로 인가된 데이터가 저장된다.

3. 중앙처리장치

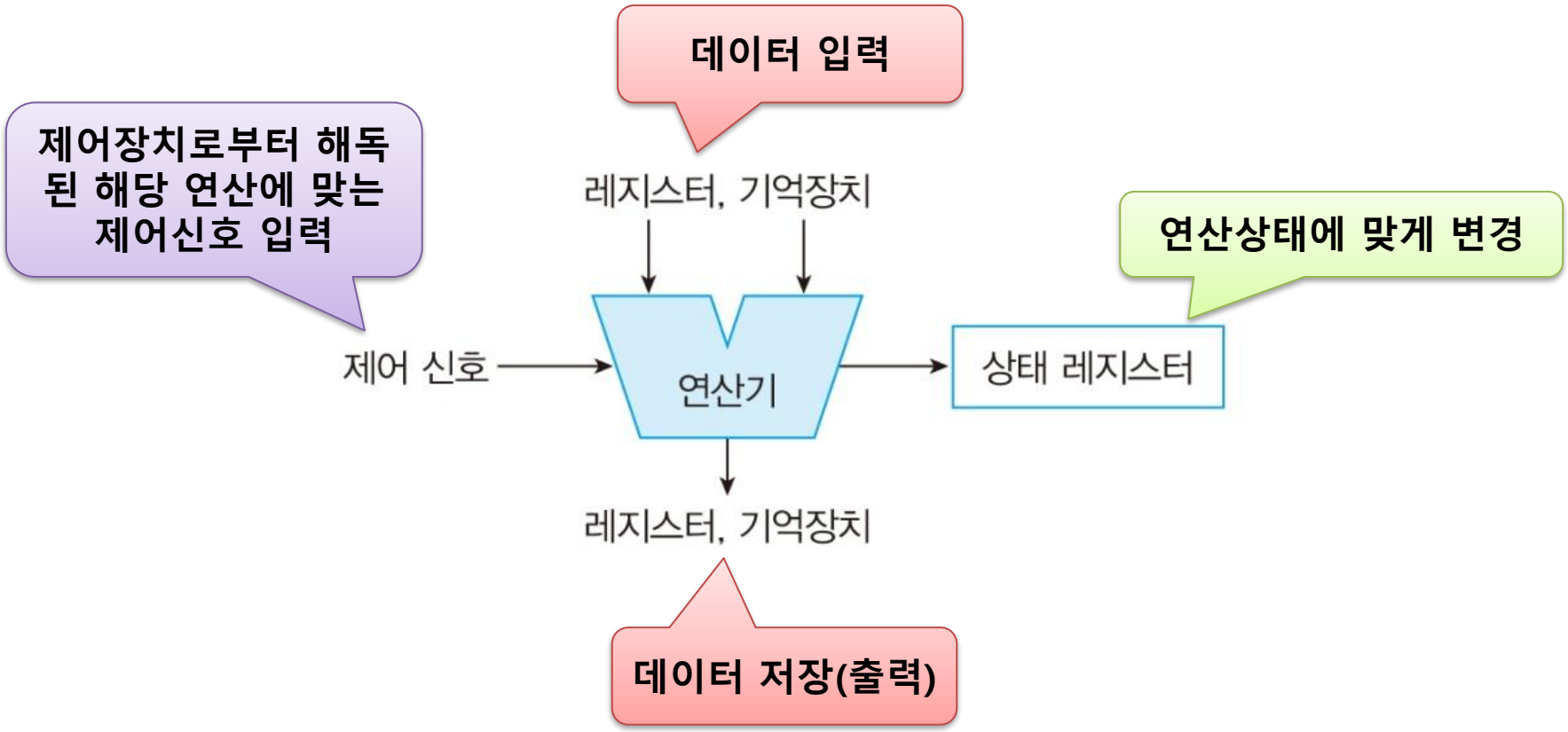
- 컴퓨터의 모든 데이터 처리와 제어 담당
  - 제어장치+연산장치+레지스터
- 중앙처리장치 구성



- 제어장치(CU)
  - 명령어 인출단계에서 명령어 레지스터 IR에 적재한 명령어를 해석하여 명령어를 실행
  - 명령어 실행단계에서 명령코드가 명령을 수행할 수 있게 필요한 제어 기능을 제공



- **연산장치(연산기, ALU)**
  - 데이터 처리 담당



4. 레지스터

- CPU 내부에 있는 초고속 기억장치
- CPU 내부에서 연산에 필요한 데이터, 연산의 중간 결과, 명령어, 주소등을 일시적으로 기억하는 임시 기억장치
- 플립플롭(Flip-Flop) 회로로 구성 : 1비트 기억소자
  - 논리회로 진도시 설명

- 용도에 따라 구분
  - 제어용 레지스터
    - 프로그램 실행 순서를 결정하는데 사용되는 레지스터
    - PC, IR, MAR, MBR 등
  - 명령어 실행용 레지스터
    - 어셈블리어 프로그래머가 프로그램 작성시 사용 가능
    - DR, AR 등

- **프로그램 카운터 레지스터(PC)**
  - 다음에 실행할 명령어의 주소 저장
  - 다음에 인출(fetch)할 명령어가 기억되어 있는 주기억장치의 번지를 기억하는 레지스터
  - 각 명령어가 인출된 후에는 자동적으로 일정 크기(워드 기준)만큼 증가(다음에 실행할 명령어 주소로 바뀜)

  - 명령인출단계의 레지스터 전송문

T0 : MAR ← PC

T1 : MBR ← Mem[MAR] , PC← PC+[명령어길이]

T2 : IR ← MBR

- **명령어 레지스터(IR)**
  - 현재 실행중인 명령어 저장
  - IR에 저장된 명령은 제어장치로 전달되어 해독된다.

- **기억장치 주소 레지스터(MAR)**
  - Memory Address Register
  - 기억장치를 액세스할 주소 저장

- **기억장치 버퍼 레지스터(MBR)**
  - Memory Buffer Register
  - 기억장치를 출입하는 데이터가 잠시 저장되는 레지스터

- 데이터 읽기(주기억장치의 내용을 레지스터로) 마이크로 연산

T0 : MAR ← AR

T1 : MBR ← Mem[MAR]

T2 : DR ← MBR

- 데이터 쓰기(레지스터의 내용을 주기억장치로) 마이크로 연산

T0 : MAR ← AR

T1 : MBR ← DR

T2 : Mem[MAR] ← MBR

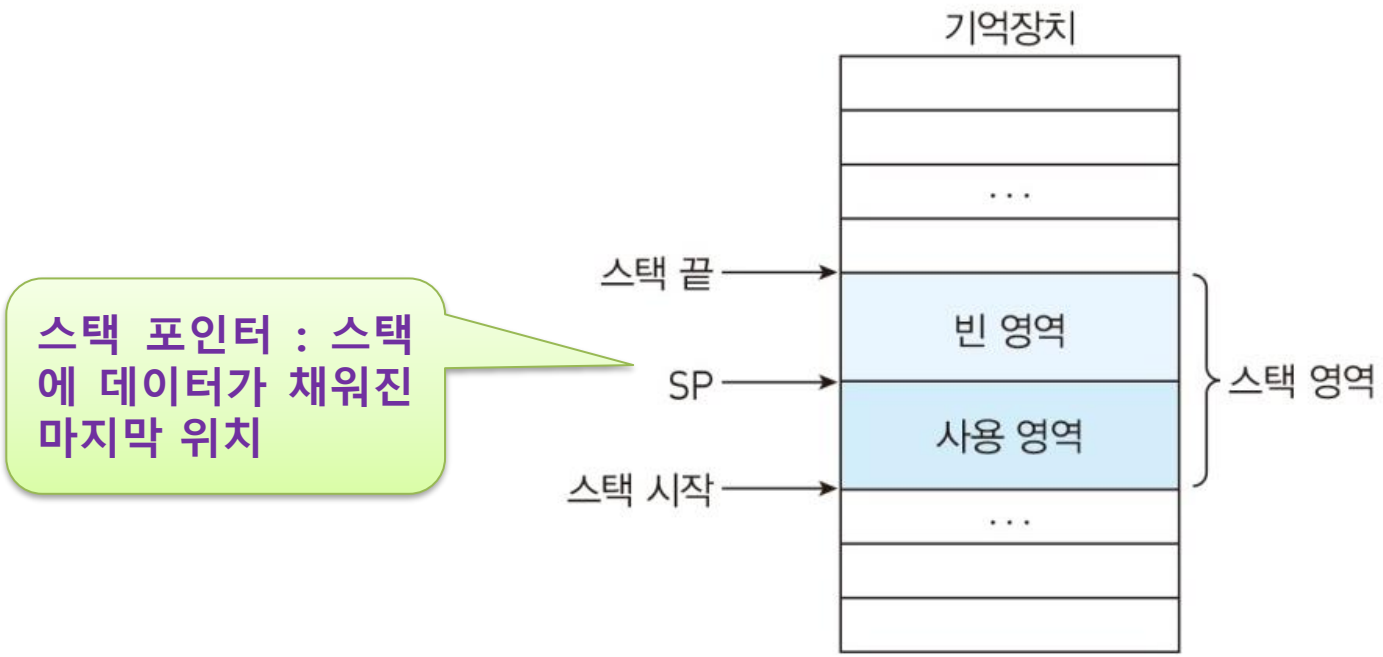
- **상태 레지스터(SR, Status Register)**
  - **플래그 레지스터**(FR, Flag Register) 또는 프로그램의 상태 즉, PSW(Program Status Word)를 저장하는 레지스터라는 의미로 **PSWR**(프로그램 상태 레지스터)이라고도 불린다.

- **누산기(AC, ACC, Accumulator)**
  - 연산의 결과를 임시로 기억하는 레지스터
  - ALU에 속하며, 연산의 중심이 된다.
  - **데이터 레지스터(DR)** 중 하나
    - 초창기 컴퓨터는 DR이 하나만 존재 = 누산기
    - 연산에 사용할 데이터를 저장하는 레지스터

- 주소 레지스터(AR, Address Register)
  - 기억장치를 액세스하기 위해 필요한 주소 저장
  - 종류 : 스택 포인터(Stack Pointer), 베이스 레지스터(Base Register), 인덱스 레지스터(Index Register) 등
- 범용 레지스터(GPR, General Purpose Register)
  - 프로그래머가 임의로 사용가능한 레지스터
  - 데이터 또는 주소 레지스터로 사용가능
  - 통상 R1, R2 등의 이름을 사용하며, RISC 프로세서에 범용 레지스터가 더 많다

- 베이스 레지스터(BR, Base Register)
  - 명령이 시작되는 최초의 번지(시작주소)를 기억하는 레지스터
- 인덱스 레지스터(IR, Index Register)
  - 명령어 실행과정에서 명령어가 지정한 번지를 수정하기 위한 레지스터

- 스택 포인터(SP, Stack Pointer)
  - 스택
    - 자료의 삽입, 삭제가 Top이라고 불리는 한쪽 방향에서만 이루어지는 자료구조
    - CPU가 기억장치의 특정영역을 스택으로 구성하여 운영
  - 후입선출 LIFO(Last In First Out) 방식



– 명령

- 자료삽입명령 Push, 스택에 저장

형식 : PUSH [operand]

//operand, 명령의 피연산자(데이터, 주소 등)

SP ← SP + [워드크기]

MAR ← SP, MBR ← 오퍼랜드

Mem[MAR] ← MBR

- 자료인출명령 Pop, 스택에서 제거

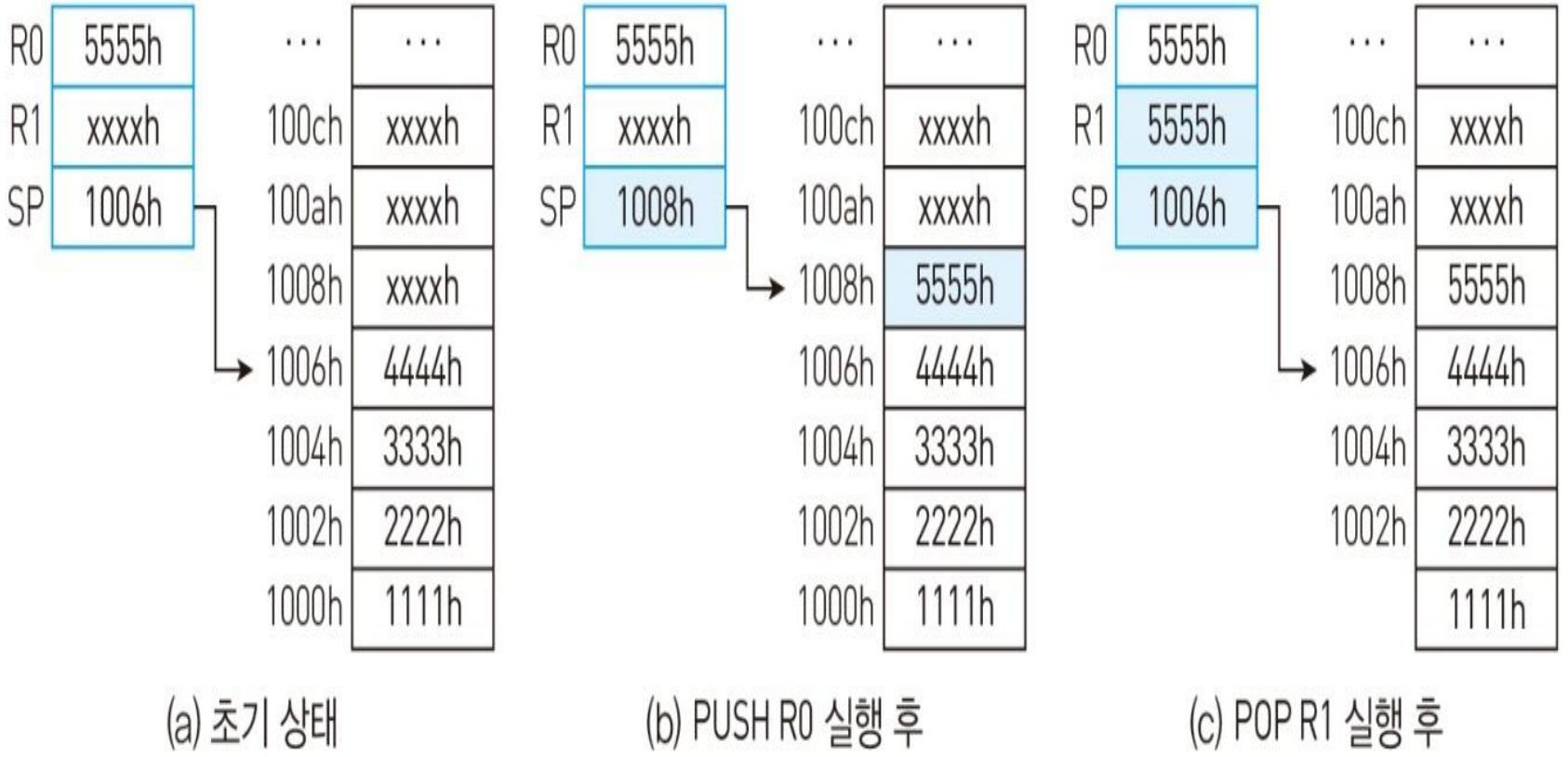
형식 : POP [operand]

MAR ← SP

MBR ← Mem[MAR]

SP ← SP - [워드크기], 오퍼랜드 ← MBR

– 스택 동작

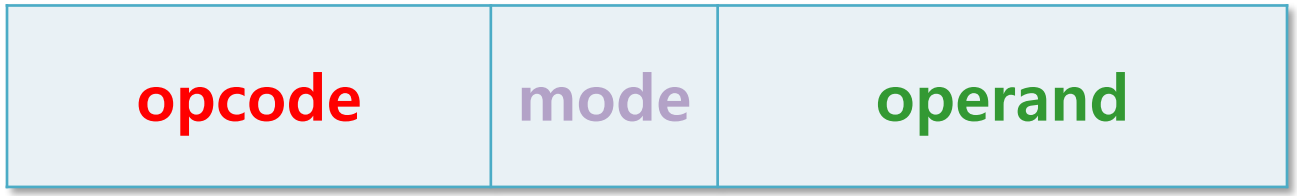

R0 5555h
R1 xxxxh
SP 1006h
... ...
100ch xxxxh
100ah xxxxh
1008h xxxxh
1006h 4444h
1004h 3333h
1002h 2222h
1000h 1111h
(a) 초기 상태
R0 5555h
R1 xxxxh
SP 1008h
... ...
100ch xxxxh
100ah xxxxh
1008h 5555h
1006h 4444h
1004h 3333h
1002h 2222h
1000h 1111h
(b) PUSH R0 실행 후
R0 5555h
R1 5555h
SP 1006h
... ...
100ch xxxxh
100ah xxxxh
1008h 5555h
1006h 4444h
1004h 3333h
1002h 2222h
1111h
(c) POP R1 실행 후

5. 명령어 집합

- **명령어 집합(instruction set)**
  - 중앙처리장치가 처리할 수 있는 명령어들의 모임
- **명령어 집합 설계를 위해 결정되어야 할 사항들**
  - 연산 종류(operation repertoire, 7. 명령의 종류)
  - 데이터 형태(data type)
  - 명령어 형식(instruction format): 명령어의 길이, 오퍼랜드 필드들의 수와 길이 등
  - 주소지정 방식(addressing mode): 오퍼랜드의 주소를 지정하는 방식

6. 명령어 구성

- 명령어의 구성요소



- 동작코드(operation code)
  - 중앙처리장치가 실행해야 할 동작을 이진수로 표현한 코드
- 오퍼랜드(operand)
  - 중앙처리장치가 동작을 실행할 대상
  - 명령어 종류에 따라 오퍼랜드 필드의 수가 다르다.

- 동작코드(opcode, 연산자부, 연산코드)
  - 연산자부의 크기는 표현할 수 있는 명령의 종류를 나타낸다
  - N Bits로 구성되면 2^N개 명령어 사용 가능
  - 명령어 개수와 연관

- 오퍼랜드(operand, 자료부, 주소부, 피연산자부)
  - 소스 오퍼랜드(source operand)와 목적지 오퍼랜드(destination operand)
    - 소스 오퍼랜드 : 동작코드가 처리할 대상
    - 목적지 오퍼랜드 : 처리한 결과를 저장할 장소
  - N Bits로 구성되면 2^N개 기억장소 주소 지정 가능
  - 기억장치 주소, 용량과 연관

- 명령어 표현 예

```
ADD  R1  A
MOV  R1  R2
ADD  C  R1  R2
```

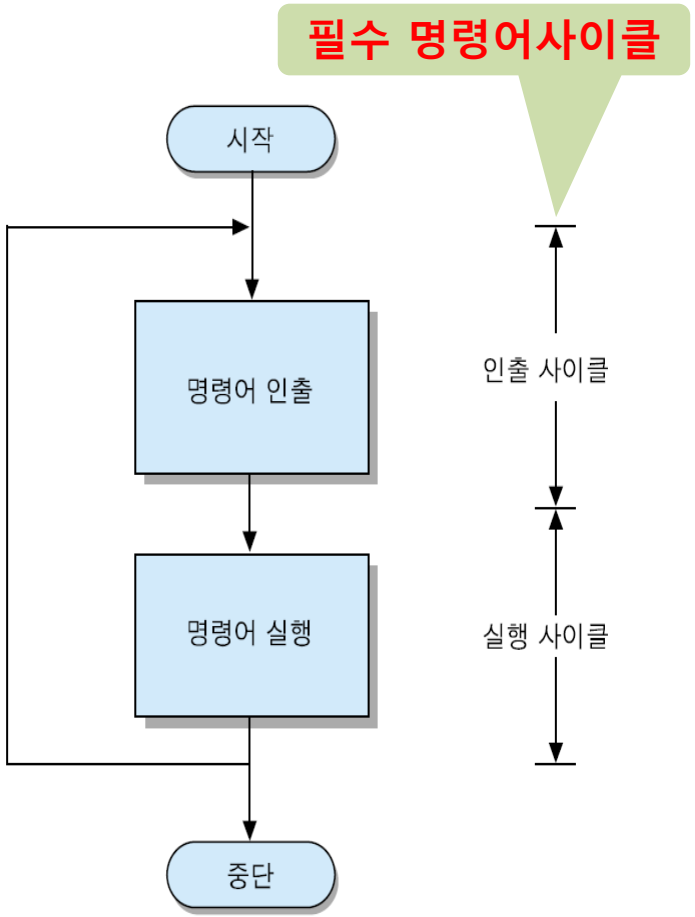
- 동작코드
  - 명령어들마다 서로 다른 2진수 코드가 주어짐
  - 니모닉 코드(mnemonic code)
    - 이진수 대신에 명령어의 의미를 쉽게 기억할 수 있도록 부여한 문자열
    - ADD, SUB, MUL, DIV, BR, PUSH, POP…..
- 오퍼랜드 종류
  - 즉치데이터(실제 데이터)
  - 레지스터 이름 = 레지스터 번호
  - 변수 이름 = 주기억장치 주소
  - 입출력 포트

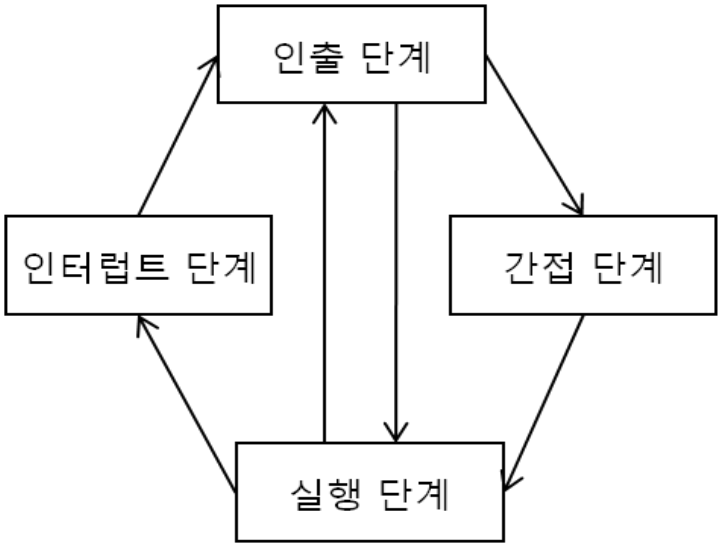
7. 명령어 종류

- 데이터처리기능(=연산기능, 함수연산기능)
  - 데이터 값을 조작해서 처리결과를 다른 장소에 저장하는 기능
  - 산술연산(ADD, SUB등), 논리연산(AND, OR등)
- 데이터전달기능
  - 컴퓨터 구성 요소간에 자료를 전달(이동)하는 기능
  - Load(적재, M→R), Store(저장, R→M), Move(이동) 등
- 제어기능
  - 명령의 실행순서를 변경
  - 분기(무조건/조건), 서브루틴호출 등
  - Goto, JMP, IF, call 등
- 입출력기능
  - 외부장치들로부터 CPU 또는 주기억장치로 데이터 이동
  - INPUT(INP), OUTPUT(OUT) 등

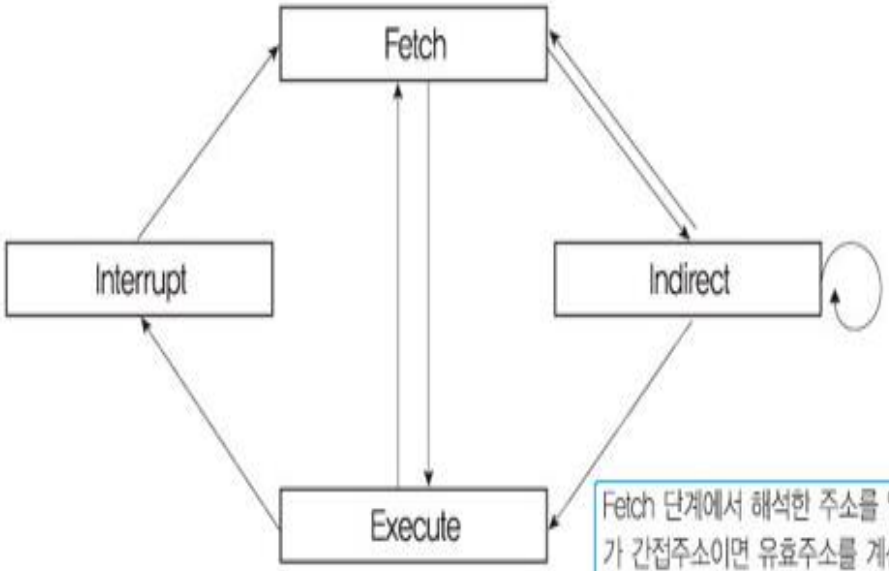
8. 명령어사이클

- 명령어사이클(instruction cycle)
  - 중앙처리장치가 한 개의 명령어를 실행하는 과정
  - 컴퓨터는 프로그램에 포함되어 있는 명령어들을 하나씩 중앙처리장치로 가져와 해독하고 실행하는 과정을 반복한다.
  - Machine Cycle, Major Cycle 이라고도 한다.
  - 각 사이클 단계에서의 CPU 현재상태정보를 **메이저 스테이트(Major State)**라고 한다. 즉 인출사이클에서 CPU는 인출상태에 있다.





명령어 사이클 = 메이저 스테이트



Fetch 단계에서 해석한 주소를 읽어온 후 그 주소가 간접주소이면 유효주소를 계산하기 위해 다시 Indirect 단계를 수행한다.

- **인출사이클**
  - 명령어를 주기억장치에서 중앙처리장치의 명령 레지스터로 가져와 해독하는 단계
  - 인출단계의 마이크로 오퍼레이션

```
T0 : MAR ← PC

T1 : MBR ← M[MAR]
     PC ← PC+1          //워드크기만큼 증가

T2 : IR ← MBR[OP]     // IR ← MBR 표현(간접 오른쪽 코드연결)
     I ← MBR[I]       // I 는 명령중 모드비트값을 나타냄
                      // 즉, 모드비트값을 I 플립플롭으로 보냄
```

- **간접사이클**
  - 인출단계에서 해석된 명령의 오퍼랜드부가 간접주소일 경우 유효주소(실제 데이터가 있는 주소)를 구하는 단계
  - 간접단계의 마이크로 오퍼레이션

T0 : MAR ← MBR[ADDR]

T1 : MBR ← M[MAR]

T2 : 동작없음

// 실행단계로 진행

T0 : MAR ← IR[ADDR]

T1 : MBR ← M[MAR]

T2 : IR[ADDR] ← MBR

// 실행단계로 진행

  - 인출된 명령어의 주소 필드 내용을 이용하여 기억장치로부터 데이터의 실제 주소를 인출하여 IR의 주소 필드에 저장

- **인터럽트사이클**
  - 인터럽트(interrupt) : 프로그램 실행 중에 CPU의 현재 처리 순서를 중단시키고 다른 동작을 수행하도록 요구하는 시스템 동작
  - 인터럽트 발생 시 복귀주소를 저장시키고, 제어순서를 인터럽트 처리 프로그램의 첫번째 명령으로 옮기는 단계
  - 인터럽트단계의 마이크로 오퍼레이션

T0 : MBR[AD] ← PC, PC ← 0

T1 : MAR ← PC , PC ← PC + 1

T2 : M[MAR] ← MBR, IEN ← 0

T0 : MBR[AD] ← PC

T1 : MAR ← SP , PC ← ISR addr

T2 : M[MAR] ← MBR, IEN ← 0

- **실행사이클**
  - CPU는 실행 사이클 동안에 명령어 코드를 해독(decode)하고, 그 결과에 따라 필요한 연산들을 수행
  - 실행단계의 마이크로 오퍼레이션은 명령어마다 다르다.