제2장 CPU의 구조와 기능 (3-1)

- 명령어 세트
  - 연산자 종류와 이에 따른 명령어 형식
- 명령어 형식 (Instruction Format)
  - 명령어 종류, 데이터 형태와 길이

2.4 명령어 세트

- 명령어 설계는 ?
- CPU 기능은 명령어들에 의해 결정됨.
- 여기서 명령어란?
  - 기계 명령어(machine instruction)
    - 또는 어셈블리 명령어(assembly instruction)

2.4 명령어 세트

- CPU에 정의된 명령어들의 집합을
  - 명령어 세트(Instruction Set)
- *명령어 세트 설계*를 위한 결정 사항
  - **연산의 종류** : CPU가 수행할 연산자 (operation)
    - 연산자 수와 종류, 명령어의 길이
  - **데이터 형태** : 연산에 사용할 데이터 형태(operand)
    - 데이터 길이, 수 표현 → 정수, 부동 소수점 수
  - **명령어 형식** : 명령어 길이, operand field들의 개수,길이
  - **주소지정 방식** : operand의 주소를 지정하는 방식

2.4.1 연산자 종류

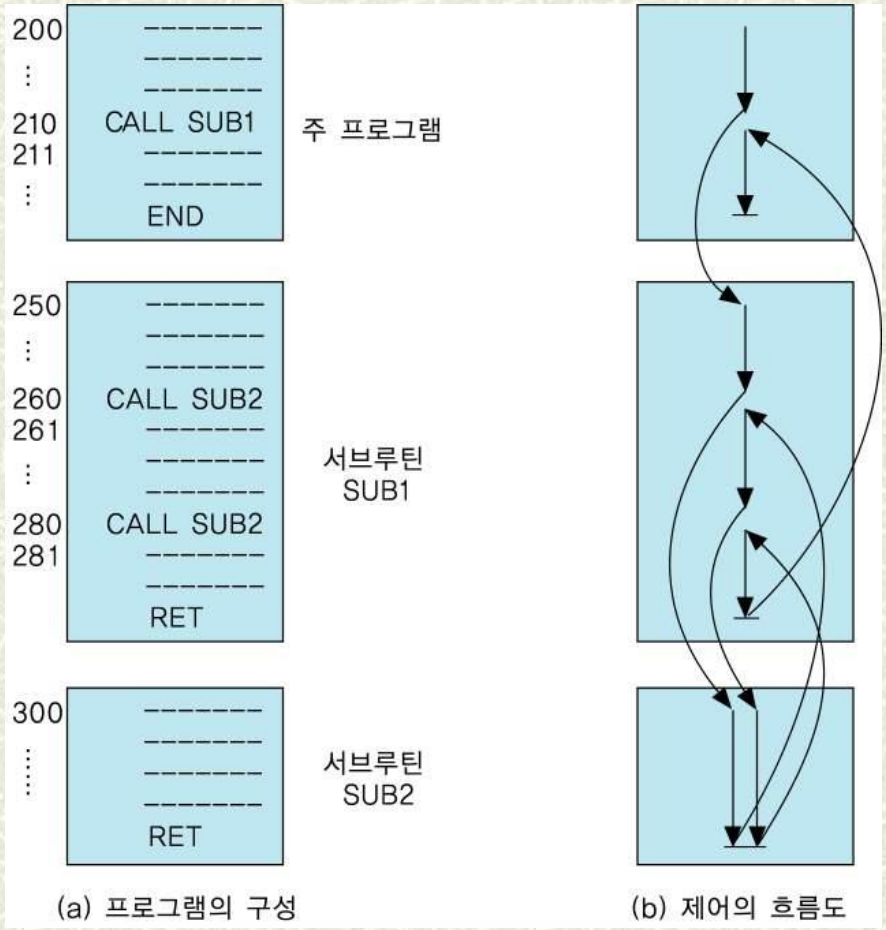
- **Operation Code** 부분은
  - +,−,x,/, Shift, Complement, Logic 연산등과 같은 동작을 정의하는 비트로 구성
    - 이 부분의 크기가 n 비트이면, 최대 2^n 개의 서로 다른 동작을 실행 가능

연산자 기능 분류

- **연산의 종류**
  - 산술 연산, 논리 연산
    - 덧셈, Shift, Complement, AND, OR, NOT
  - 데이터 전송기능
    - load, store, move
      - 레지스터-메모리간, 또는 레지스터간의 데이터 전송
  - I/O 기능 --- 자세한 설명 7장
    - 외부 장치 입출력(in, out)
  - 프로그램 제어 기능
    - 시스템 제어, 프로그램 제어(if, jump(goto), call)

연산자 기능 분류(계속)

- 프로그램 제어 기능*(계속)*
  - 프로그램에서의 subroutine call 또는 interrupt 서비스 처리
    - 예) Subroutine Call

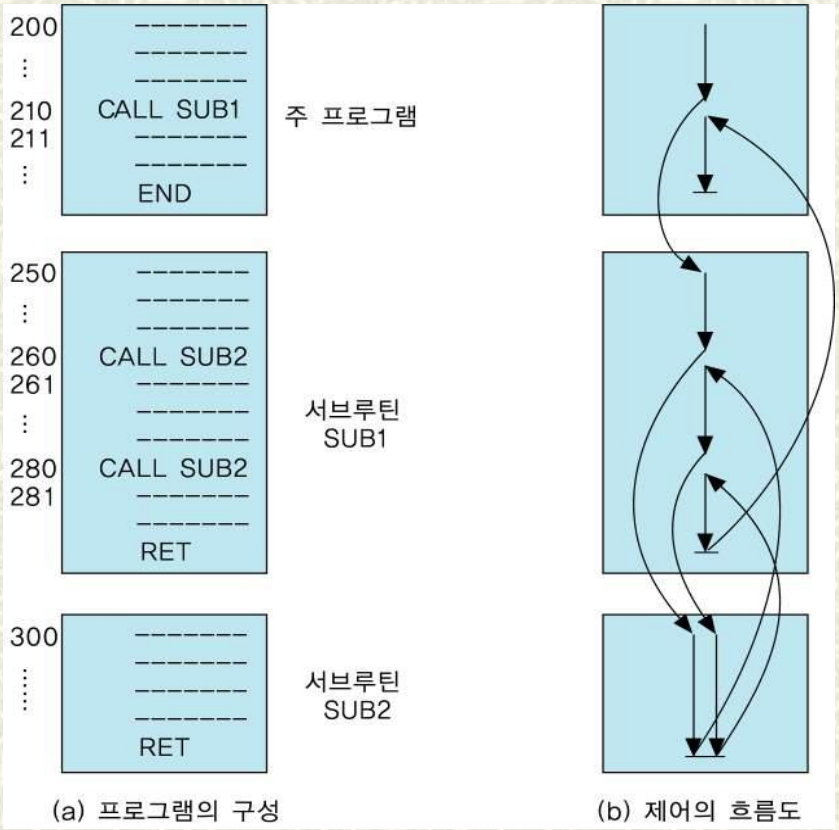


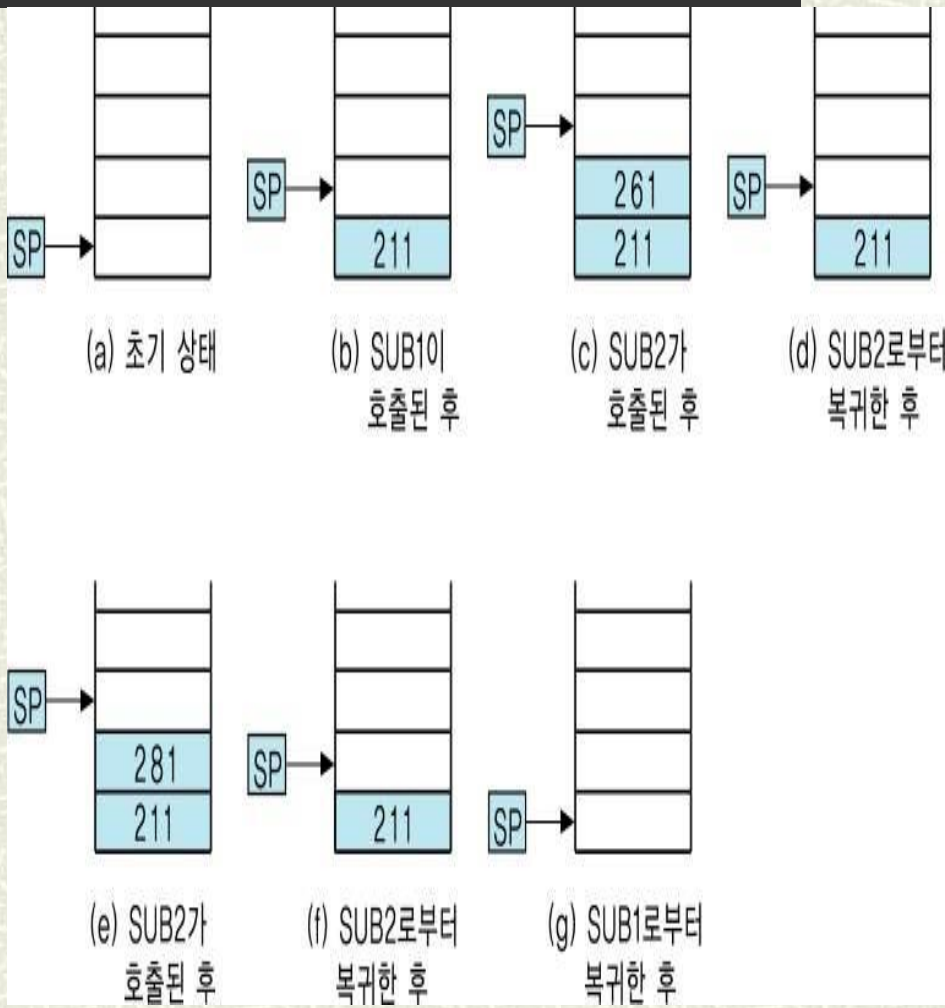
Call X

t_0 : MBR ← PC
t_1 : MAR ← SP, PC ← X
t_2 : M[MAR] ← MBR, SP ← SP − 1

Return

t_0 : SP ← SP + 1
t_1 : MAR ← SP
t_2 : PC ← M[MAR]





2.4.2 명령어 형식(Instruction format)

- 명령어는 연산자(operation)와 그 연산자에 사용되는 자료(operand)로 구성됨.
  - **operation code** : 수행될 연산(LOAD, ADD등)
  - **operand** : 수행에 필요한 데이터 or 데이터 주소

2.4.2 명령어 형식(Instruction format)

- 명령어의 설계 과정시 필수 고려사항
  - 컴퓨터에 어떤 종류의 연산자들을 사용할 것인가?
    - 연산자 종류
  - 자료를 어떻게 표현할 것인가?
    - 데이터 길이, 형식
    - 주소지정방식
- 위의 두가지를 어떠한 형태로 모아서 명령을 형성하여 실행할 것인가?

명령어 형식의 종류

- 0-주소(無 번지) 명령 형식
  - 번지 필드가 필요 없고, 동작코드 만 존재함
    - Stack 구조의 컴퓨터에서 필요 (ADD, MUL 등)
    - 단 , PUSH, POP 명령에서는 하나의 번지 필요

- 1-주소 명령 형식
  - 데이터 처리가 Accumulator(누산기)에서 이루어짐. 결과도 누산기에 저장

명령어 형식의 종류 (계속)

- 2-주소 명령 형식
  - 실제 많이 사용됨
  - 레지스터나 메모리를 지정하는 두개의 번지 필드를 갖는다.(그림 2-21참조)

Operation	Operand 1	Operand 2

- 3-주소 명령 형식
  - 레지스터나 메모리를 지정하기 위한 두개의 주소 필드와 연산 결과를 저장하기 위한 한개의 주소 필드 필요

Operation	Operand 1	Operand 2	Operand 3

예) x=(A+B) * (C-D) 계산

- *2-주소 방식*
  - MOV R1, A
  - ADD R1, B
  - MOV R2, C
  - SUB R2, D
  - MUL R1, R2
  - MOV X, R1
- *3-주소 방식*
  - ADD R1, A, B
  - SUB R2, C, D
  - MUL X, R1, R2

산술식 Z = A × (B + C)

0-주소명령	1-주소명령	2-주소명령	3-주소명령

PUSH B
PUSH C
ADD
PUSH A
MUL
POP Z
	LOAD B
	ADD C
	MUL A
	STORE Z
		MOV R1, B
		ADD R1, C
		MUL R1, A
		MOV Z, R1
			ADD R1, B, C
			MUL Z, A, R1

제2장 CPU의 구조와 기능 (3-2)

- 명령어 세트
  - 연산자 종류와 이에 따른 명령어 형식
- 명령어 형식 (Instruction Format)
  - 명령어 종류, 데이터 형태와 길이
- 주소 지정 모드(Addressing Mode)
  - 오퍼랜드의 유효 주소 결정 방법
  - (실습) 주소 지정 모드

2.4.3 주소지정모드(Addressing Mode)

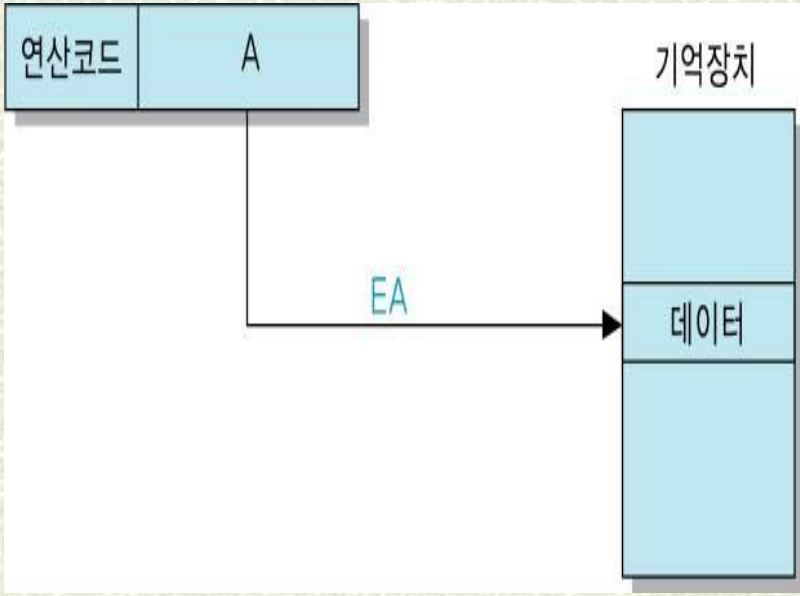
실제 데이터가 저장된 장소의 주소(EA)를 지정하는 방법

- 묵시적(implied mode)
  - operand 필드가 필요 없음
    - INC, CLC, Shift
    - (장점) 명령어 길이가 짧다
- 직접값(즉치) 모드(immediate mode)
  - 번지(operand)필드에 data가 직접 있음
    - MOV A, 50
    - 데이터를 인출하기 위한 기억장치 접근 불필요
      - (장) 빠른 실행 싸이클
    - (단점) 사용 데이터 크기가 작게 됨.

주소 지정 모드 (Addressing Mode) (계속)

데이터가 저장된 기억장치의 실제 주소

- **직접 주소 모드** *(Direct addressing mode)*
  - EA = A
  - 오퍼랜드의 저장주소(유효주소)를 명령어의 오퍼랜드 필드에 직접 지정함.
  - ADD A
  - (단) 주소공간이 operand 필드 비트 수에 의해 제한됨.

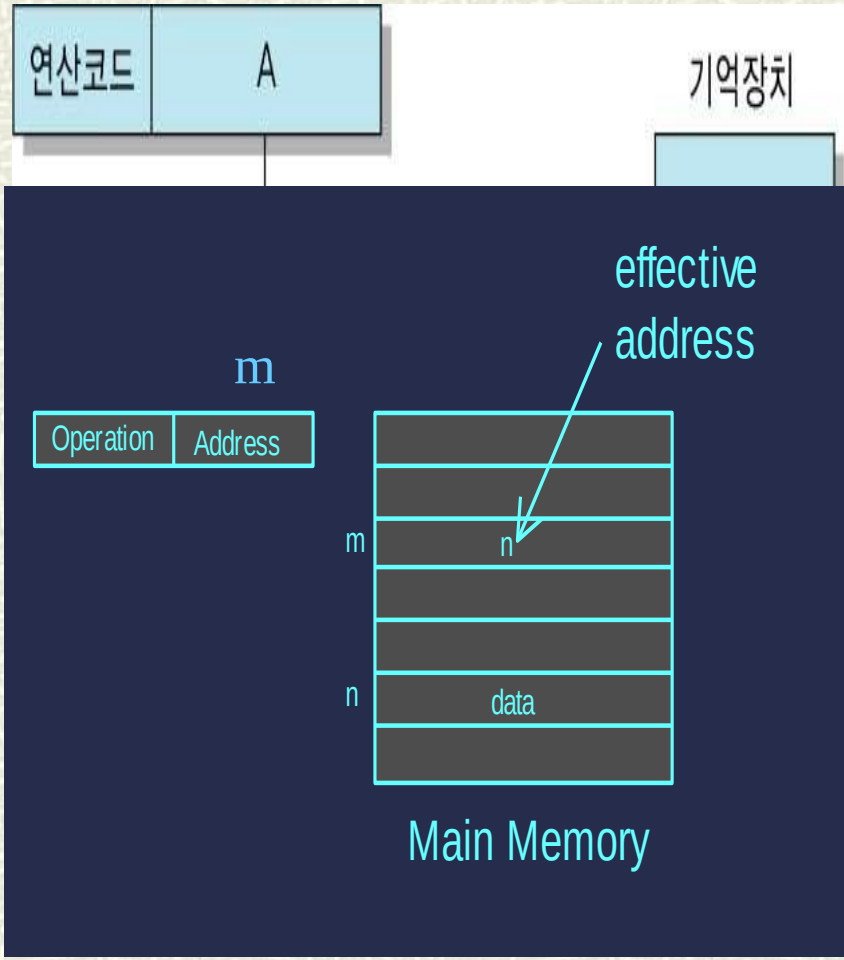


주소 지정 모드 (Addressing Mode)

(계속)

간접주소 모드 *(Indirect addressing mode)*

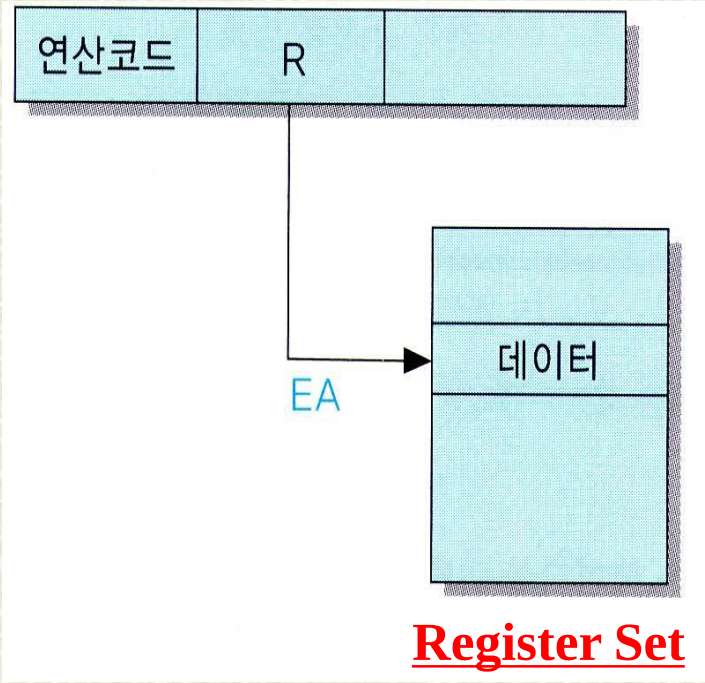
- EA = (A)
- 실제 값이 저장된 **주소**(유효주소)가 명령어의 오퍼랜드 필드에 나타남.
  - MOV R1, @M[100]
- 명령어의 오퍼랜드 필드에 있는 주소는 실제 데이터를 갖는 유효주소 저장



주소 지정 모드 (Addressing Mode)

(계속)

- **레지스터 주소 모드(*Register addressing mode*)**
  - 오퍼랜드가 레지스터에 저장
    - EA = R
    - 예) ADD R1, R2
  - (장) 데이타를 위해 메모리 접근 불필요
    - 명령어 실행이 빨라짐.
  - (단) 레지스터의 갯수가 제한 적임.

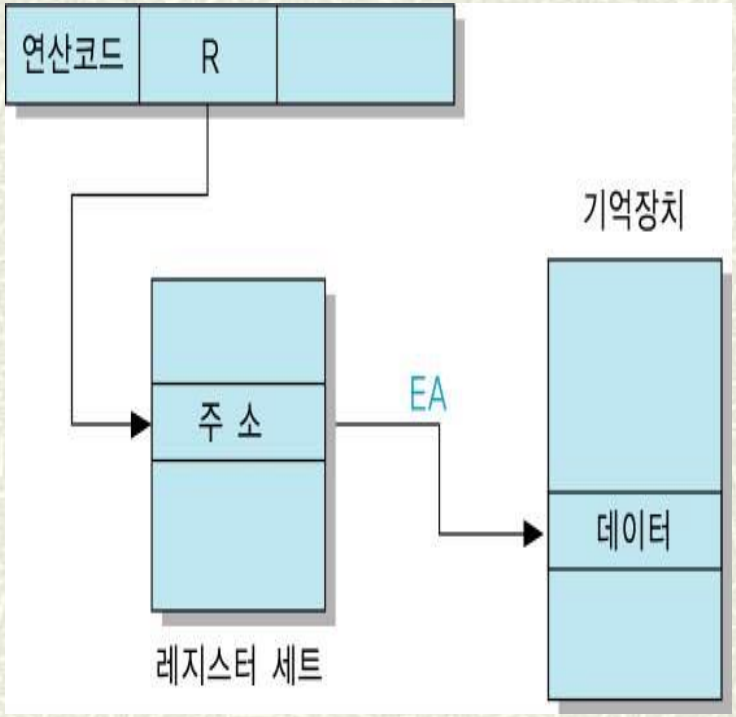


주소 지정 모드 (Addressing Mode)

(계속)

레지스터 간접주소 모드 *(Register-indirect addressing mode)*

- EA=(R)
- 주소지정 범위는 레지스터 크기에 비례함.
  - 레지스터 길이 16bit 라면 ?
    - 64K개 Byte word 지정 가능
  - 레지스터 길이 32bit 라면 ?
    - ? 개 Byte word 지정 가능

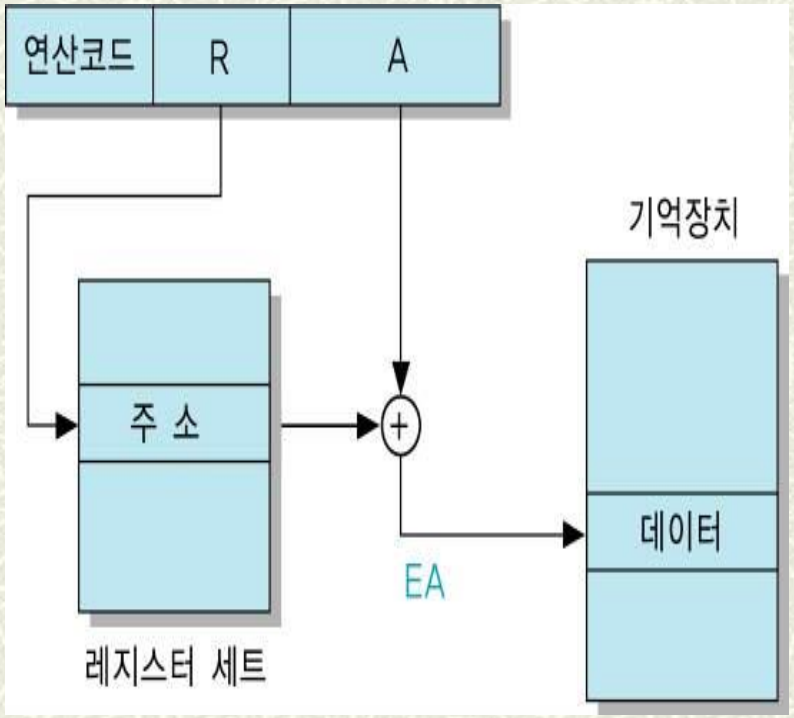


주소 지정 모드 (Addressing Mode)

(계속)

변위 주소지정 모드 *(Displacement addressing mode)* EA = A + (R)

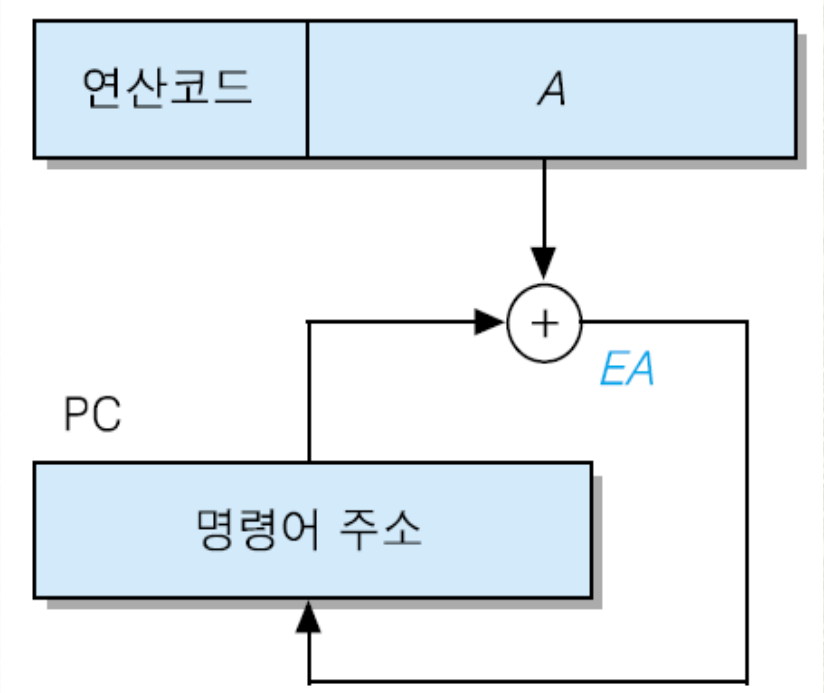
- 직접 주소지정 방식과 레지스터 간접주소방식의 혼합
  - ① 상대주소 방식 (Relative addressing mode)
  - ① 인덱스 주소 방식 (Index addressing mode)
  - ① 베이스 레지스터 주소 방식 (Base addressing mode)



주소 지정 모드 (Addressing Mode)
Displacement addressing mode(계속)

상대 주소 모드 (Relative Addressing Mode)

- EA=A+(PC)
  - 실제 값이 저장된 유효주소가 ***현재실행중인 명령어의 위치에 근접한 경우*** 사용
    - 유효주소
      - = 명령어의 주소부분 + (PC)
- ✲ 주로 ***분기명령어***에 사용됨



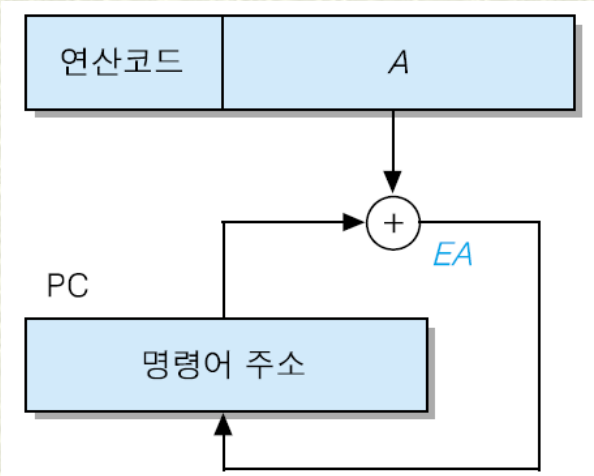
상대 주소지정 방식 (relative addressing mode)

- 프로 그램카운터(PC)를 레지스터로 사용하여 EA를 계산

$$EA = A + (PC)$$

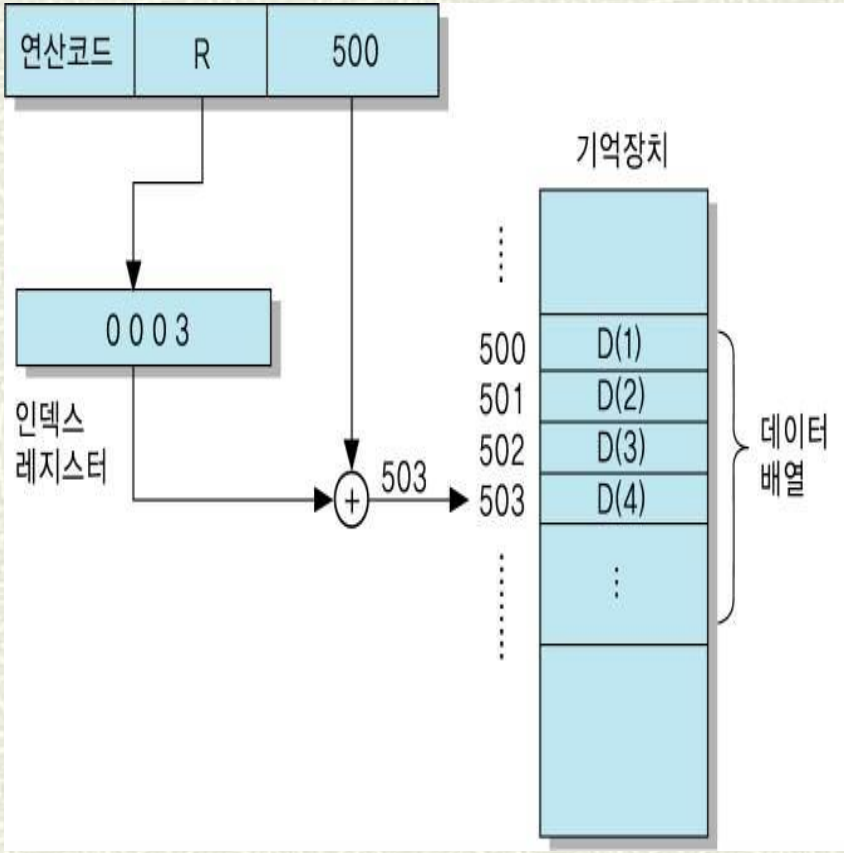
단, A는 2의 보수

- 주로 분기 명령어에서 사용
  - $A > 0$: 앞(forward) 방향으로 분기
  - $A < 0$: 뒷(backward) 방향으로 분기

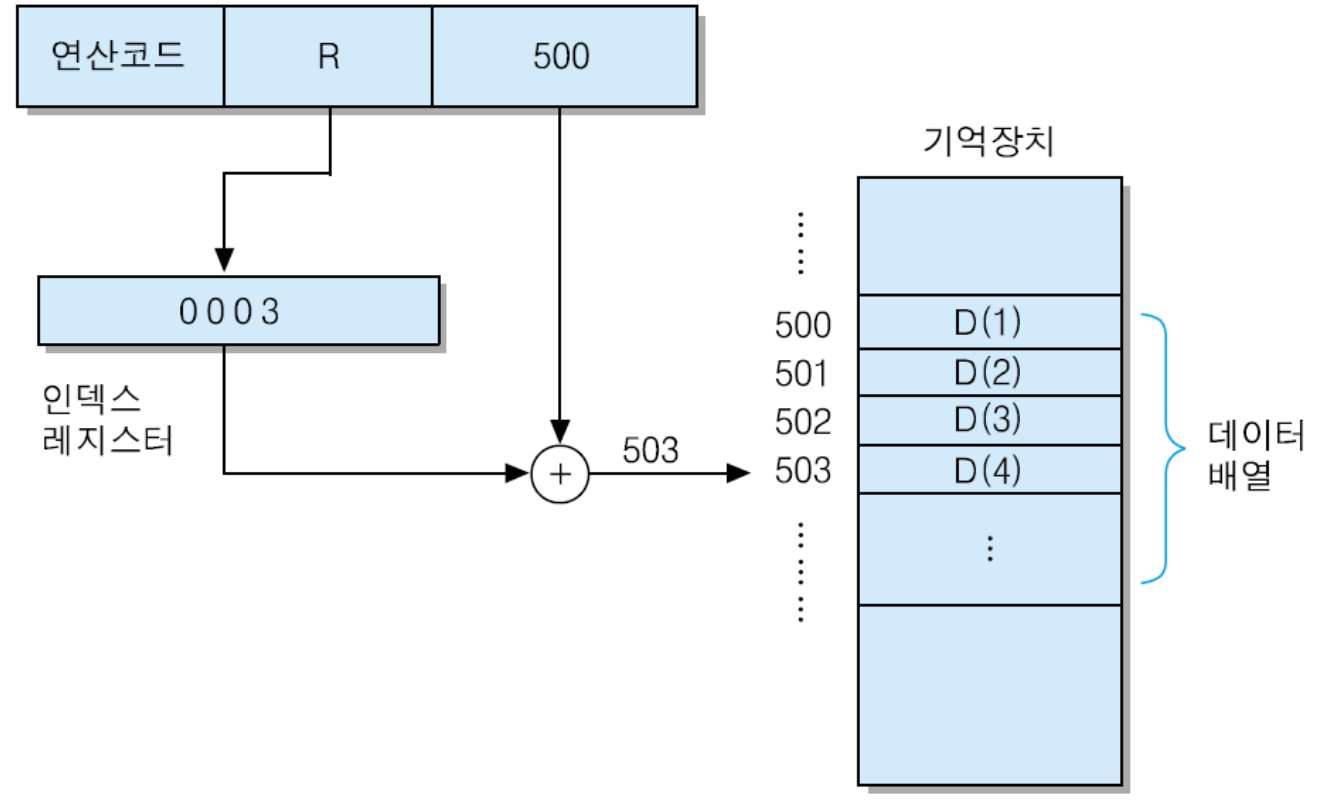


주소 지정 모드 (Addressing Mode)
Displacement addressing mode(계속)

- *인덱스 주소지정 방식 (Indexed addressing mode)*
  - EA=A + (IX)
  - 인덱스 레지스터 내용(IX)와 변위 A를 더하여 유효 주소 결정
  - 연속된 데이타를 차례대로 액세스 용이
    - *배열(array) 내용 접근*



[예] 데이터 배열이 기억장치 500 번지부터 저장, 명령어 주소 필드에 '500' 이 포함되어 있을 때, 인덱스 레지스터의 내용 (IX) = 3 이라면 → 데이터 배열의 네 번째 데이터 액세스



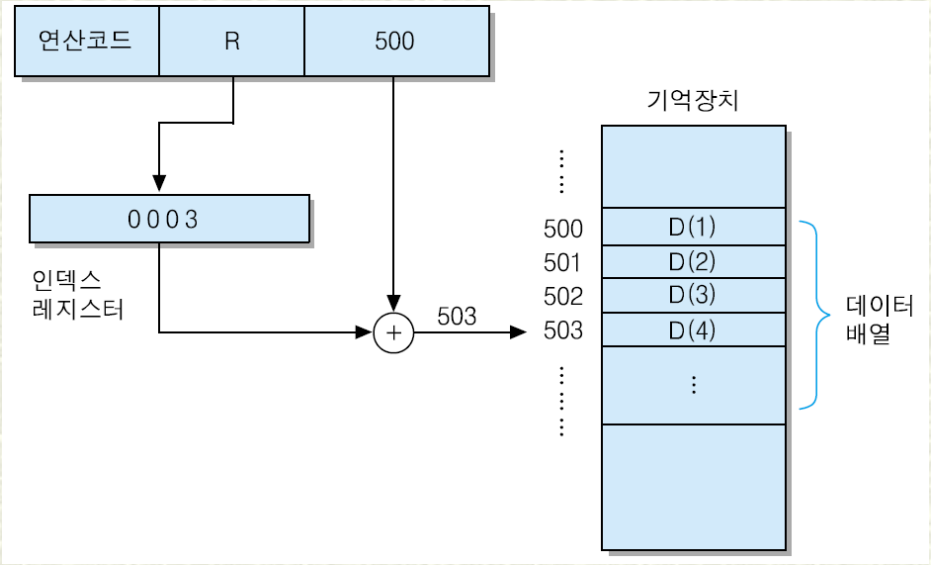
인덱스 주소지정 방식 (계속)

자동 인덱싱(auto-indexing)

- 명령어가 실행될 때마다 인덱스 레지스터의 내용이 자동적으로 증가 혹은 감소
- 이 방식이 사용된 명령어가 실행되면 아래의 두 연산이 연속적으로 수행됨

$$EA = (IX) + A$$

$$IX \leftarrow IX + 1$$



주소 지정 모드 (Addressing Mode)

Displacement addressing mode(계속)

베이스 -레지스터 주소 지정모드(Base-register addressing mode) EA = A + (BR)

- 베이스 레지스터에는 기준 주소 저장
  - 베이스 레지스터의 내용과 변위 A를 더하여 유효 주소를 결정
- 프로그램의 위치를 지정하는데 사용
- ❖ 다중 프로그램밍 *시스템에서 프로그램과 데이타들을 다른 위치로 이동하는데 효과적.*
  - 프로그램내의 주소 필드, 분기 주소를 변경 필요 없음 (.com : .exe)

2.4.4 실제 상용 프로세서들의 명령어 형식

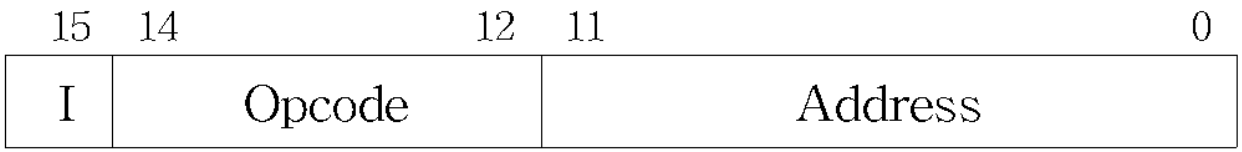
- CISC(Complex Instruction Set Computer) 프로세서
  - 명령어들의 수가 많음
  - 명령어 길이가 일정하지 않음(명령어 종류에 따라 달라짐)
  - 주소지정 방식이 매우 다양함

  [예] PDP 계열 프로세서, Intel Pentium 계열 프로세서

- RISC(Reduced Instruction Set Computer) 프로세서
  - 명령어들의 수를 최소화
  - 명령어 길이를 일정하게 고정
  - 주소지정 방식의 종류를 단순화

  [예] ATmega microcontroller, ARM 계열 프로세서

◎ Basic Computer Instruction Format



◎ Basic Computer Instruction

Symbol

Symbol	Hexadecimal Code		Description
	I=0	I=1	
AND	0xxx	8xxx	AND 연산
ADD	1xxx	9xxx	ADD 연산
LDA	2xxx	Axxx	Load 연산
STA	3xxx	Bxxx	Store 연산
BUN	4xxx	Cxxx	Branch Unconditionaly
BSA	5xxx	Dxxx	Branch & save return addr.
ISZ	6xxx	Exxx	Increment and skip if zero

0011 0001 0010 0011

1011 0001 0010 0011

1100 0010 0000 0100