CPU의 구조와 기능 (2)
(제 4주 차)

서울사이버대학교

오 창 환

학습 목표

- 연산의 종류를 설명할 수 있다.
- 명령어 형식을 설명할 수 있다.
- 주소지정 방식을 설명할 수 있다.

학습 내용

- 연산의 종류
- 명령어 형식
- 주소지정 방식

연산의 종류 (1)

• CPU가 수행할 수 있는 연산(operation)의 종류는 컴퓨터에 따라 매우 다양하며 어떤 컴퓨터든 가지고 있는 일반적인 연산들은 다음과 같음.

(1) 데이터 전송

* 레지스터와 레지스터 간, 레지스터와 기억장치 간, 혹은 기억장치와 기억장치 간에 데이터를 이동하는 동작임.

(2) 산술 연산

* 덧셈, 뺄셈, 곱셈, 나눗셈 등과 같은 기본적인 산술 연산들을 말함.

(3) 논리 연산

* 데이터의 각 비트들 간에 대한 AND, OR, NOT 및 exclusive-OR 등과 같은 논리 연산을 수행함.

(4) 입출력 I/O

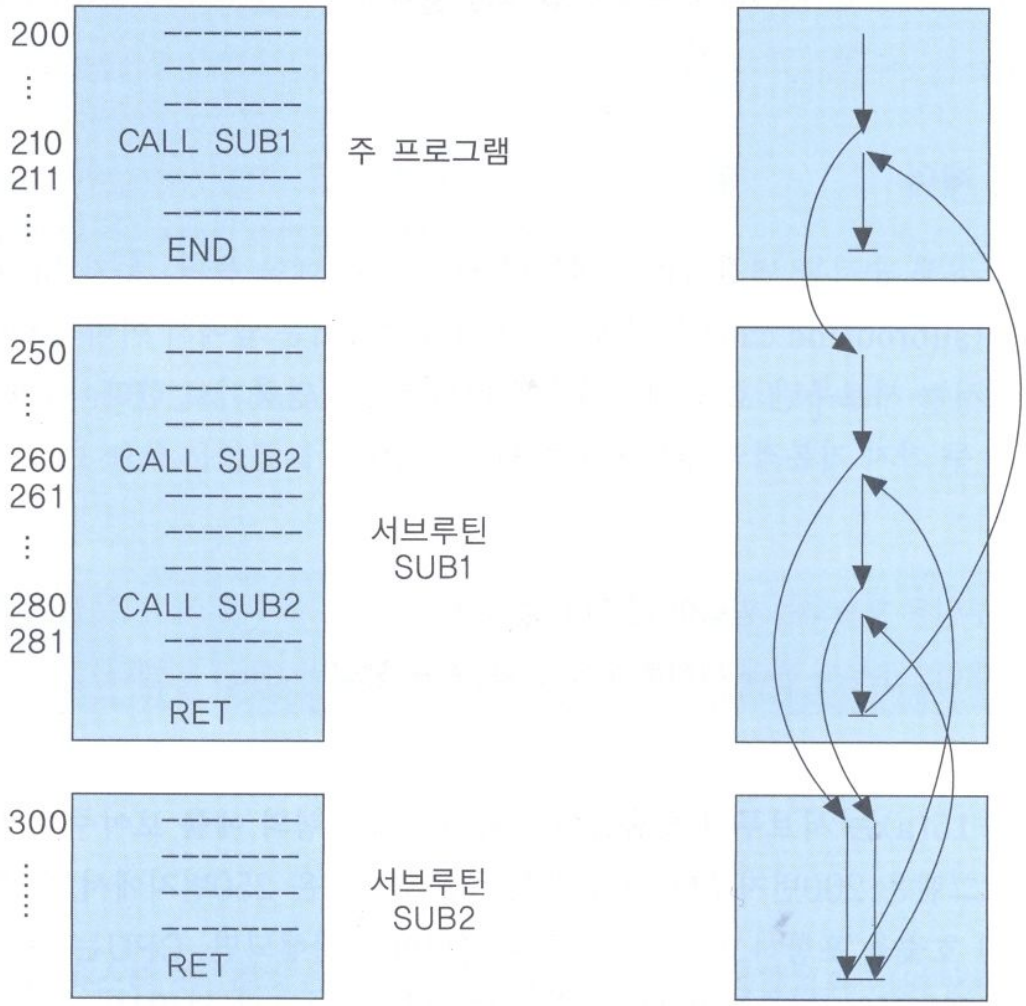
* CPU와 외부 장치들 간의 데이터 이동을 위한 동작들이 수행됨.

연산의 종류 (2)

(5) 프로그램 제어

* 명령어 실행 순서를 변경하는 연산들이 필요한데 여기에는 분기(branch)와 서브 루틴 호출(subroutine call) 등이 있음.
* 서브루틴 호출 메커니즘은 다음과 같은 두 가지 기본적인 명령어들이 필요함.
  - 서브루틴을 호출하는 명령어(CALL 명령어)
  - 서브루틴으로부터 원래 위치로 되돌아가는 복귀 명령어(RET 명령어)
* 아래 그림은 서브루틴 호출들이 포함된 프로그램의 예를 보여주고 있는데, 주프로그램은 200번지부터 시작되고, 주프로그램 안에는 250번지에서 시작되는 서브루틴(SUB1) 호출이 있음. 이 호출 명령어가 실행되면, CPU는 주프로그램의 실행을 멈추고, 250번지에서 다음 명령어를 인출함으로써 SUB1의 실행을 시작함. SUB1 내에서 300번지에 위치한 SUB2가 두 번 호출되는데, 각 호출 때마다 SUB1의 실행은 일시 중단되고, SUB2가 실행됨.
* RET 명령어는 서브루틴을 호출하기 전에 CPU가 원래 실행하던 프로그램으로 돌아가서 CALL 명령어 다음의 명령어를 계속 실행하도록 해 줌.

연산의 종류 (3)



(a) 프로그램의 구성 (b) 제어의 흐름도

- 서브루틴이 포함된 프로그램이 실행되는 순서

연산의 종류 (4)

* CALL X라는 명령어가 X번지에 있는 서브루틴을 호출하는 명령어라고 할 때에,
  이 명령어가 실행되어 서브루틴으로 분기될 때는 인터럽트의 경우와 마찬가지로
  복귀할 주소를 스택에 저장해야 함.
  복귀 주소는 이 명령어의 다음에 위치한 명령어의 주소이므로 현재의 PC 내용이며
  CALL X 명령어에 대한 마이크로-연산들은 다음과 같음.

t_0 : MBR ← PC

t_1 : MAR ← SP, PC ← X

t_2 : M[MAR] ← MBR, SP ← SP-1

* 상기에서 SP는 스택 포인터를 의미함. 결과적으로 PC 내용, 즉 서브루틴 수행이
  완료된 후에 복귀할 주소는 SP가 지정하는 스택의 최상위에 저장됨.
  여기에서 유의할 사항은 SP는 항상 스택이 비어있는 최상위 주소를 가리키므로,
  일단 복귀 주소를 그 빈 자리에 저장한 다음에는 SP의 값을 1만큼 감소시켜야 함.

연산의 종류 (5)

* 복귀 명령어인 RET의 마이크로-연산에서는 첫 번째 주기에서 SP의 내용을 1증가 시켜서 복귀 주소가 저장된 스택 위치를 가리키게 한 다음에 복귀 주소를 스택으로부터 인출하여 PC에 적재함.

t_0 : SP ← SP+1

t_1 : MAR ← SP

t_2 : PC ← M[MAR]

연산의 종류 (6)

- 아래 그림은 상기의 프로그램을 이용하여 서브루틴 호출과 복귀가 이루어지는 동안에 복귀 주소들이 스택에 저장되거나 인출되는 과정을 보여주고 있음.

  (a) 초기 상태에서 SP는 스택으로 사용되는 기억장치 영역의 가장 아래 위치를 가리키고 있음.

  (b) 서브루틴 SUB1이 호출되면서 원래의 PC 내용인 211이 스택에 저장되고, SP는 1이 감소하여 그 위의 빈 장소를 가리킴.

  (c) SUB2가 호출되면서 다시 PC의 내용인 261이 스택에 저장되며 SP는 다시 1이 감소됨.

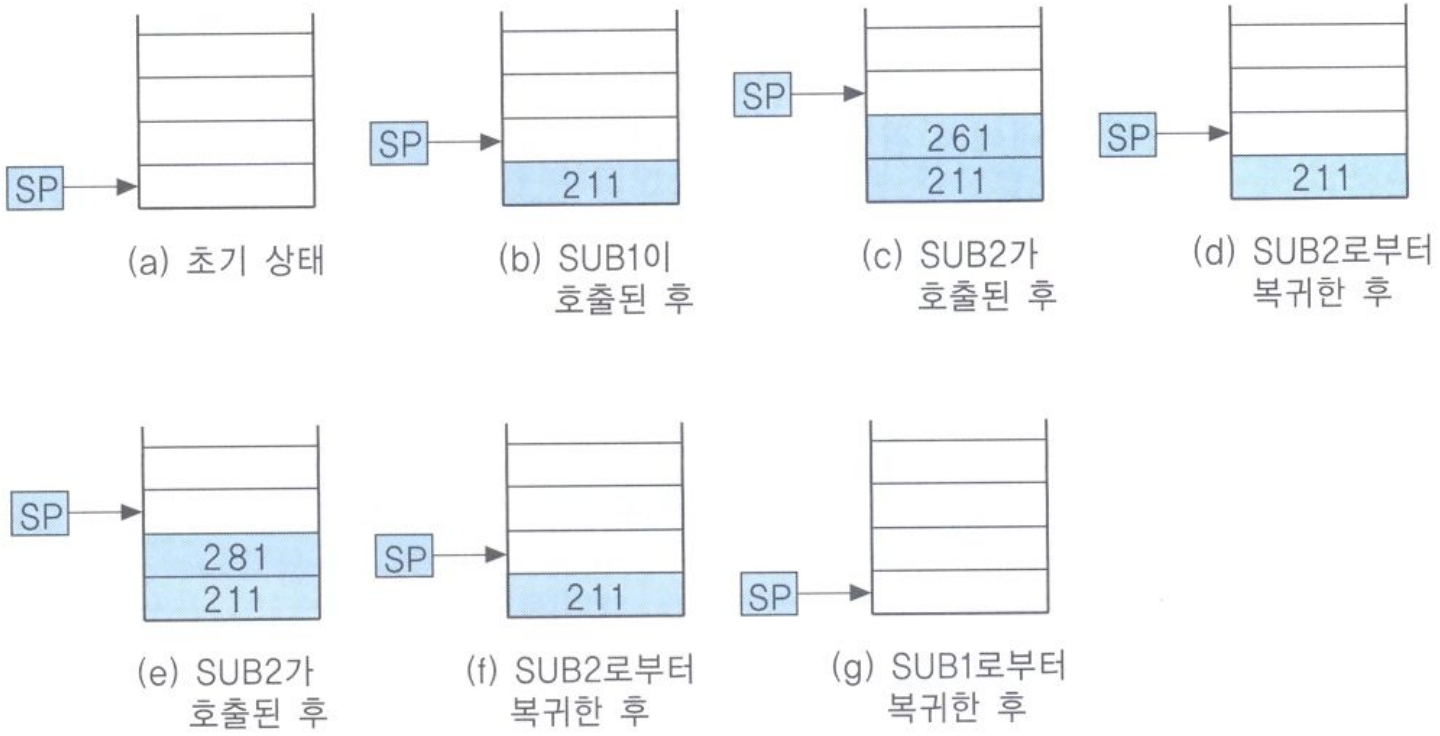
  (d) SUB2의 수행이 완료되었을 때 SUB1으로 복귀할 주소를 스택으로부터 인출함. 이것은 RET 명령어를 이용하여 이루어짐 : SP의 내용을 1 증가하고, 그 내용이 가리키는 위치로부터 복귀 주소(261)를 인출하여 PC에 적재함.

  (e) 다시 SUB2가 호출되면서 PC의 내용(281)이 스택에 저장됨.

  (f) SUB2로부터 복귀할 때, 스택으로부터 복귀 주소(281)가 인출되어 PC로 적재됨.

연산의 종류 (7)

(g) SUB1으로부터 주프로그램으로 복귀할 때, 복귀 주소(211)가 인출되어 PC로 적재됨. 스택은 비워지며, SP는 처음과 같은 값을 가지게 됨.



• 프로그램 수행 과정에서 스택에 저장되는 내용들

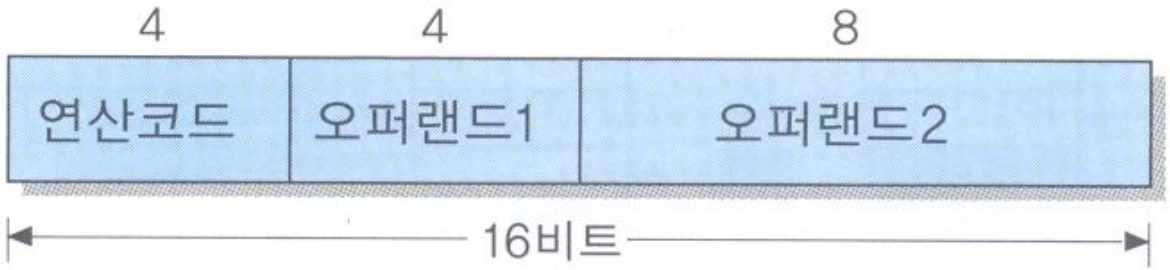
2 교시

명령어 형식 (1)

- 각 명령어는 일련의 비트들에 의해 표현되며, 이 비트들은 편의상 여러 개의 필드(field)들로 나눌 수 있는데, 필드들의 수와 배치 방식 및 각 필드의 비트 수를 명령어 형식이라고 함.
- 아래 그림의 명령어 형식에서는 연산 코드 필드에 4비트, 오퍼랜드1 필드에 4비트, 오퍼랜드2 필드에 8비트가 각각 할당되어 있음.
  연산 코드의 비트 수는 CPU가 수행할 수 있는 연산의 수를 결정해주는데, 이 필드의 길이가 4비트이므로 2의 4승 = 16가지의 연산들이 정의될 수 있음.
- 오퍼랜드 필드의 비트 수는 오퍼랜드가 다음 세 가지 중의 어떤 것인지에 따라 각각의 범위가 결정됨.
  * 데이터 : 표현 가능한 수의 크기가 결정됨.
  * 기억장치 주소 : CPU가 오퍼랜드 인출을 위하여 직접 주소를 지정할 수 있는 기억장치 영역의 범위가 결정됨.
  * 레지스터 번호 : 데이터 저장에 사용될 수 있는 레지스터의 수가 결정됨.

명령어 형식 (2)

- 예를 들어서 오퍼랜드1이 레지스터 번호를 지정하고, 오퍼랜드2가 기억장치 주소를 지정한다고 하면, 레지스터는 최대 16개까지 지정할 수 있고, 주소지정 가능한 기억장치의 주소 범위는 0번지부터 255번지까지가 됨.
- 오퍼랜드를 한 개만 필요로 하는 명령어의 경우에는 두 개의 오퍼랜드 필드를 합하여 12비트를 사용할 수 있으며, 만약 2의 보수로 표현되는 데이터라면 그 범위가 -2048부터 +2047까지가 되고, 기억장치 주소라면 2의 12승 = 4096개의 기억장치 주소를 지정할 수 있음.
- 이와 같이, 명령어를 구성하는 각 필드의 비트 수에 따라 연산의 수와 데이터의 표현 범위 혹은 기억장치 주소지정 영역 등이 결정됨.



• 16-비트 명령어 형식의 예

명령어 형식 (3)

- 명령어가 한 개, 두 개, 세 개의 오퍼랜드를 포함하는 것을 각각 1-주소 명령어 (one-address instruction), 2-주소 명령어(one-address instruction), 3-주소 명령어(one-address instruction) 라고 함.
- 1-주소 명령어

  ADD X ; AC ← AC + M[X]

  이 명령어는 '기억장치 X번지의 내용과 누산기(AC)의 내용을 더하고, 그 결과를 다시 누산기에 저장하라.'는 명령어임.

  즉 연산에 사용되는 두 개의 오퍼랜드들 중의 하나는 누산기의 내용이 되고, 결과값도 항상 누산기에 저장되도록 되어 있어서, 명령어 코드는 주소 X만 포함하면 되므로 오퍼랜드 필드의 모든 비트들이 주소지정에 사용될 수 있어서 더 넓은 영역의 주소들을 지정할 수 있게 됨.

명령어 형식 (4)

- 2-주소 명령어

  ADD R1, R2 ; R1 ← R1 + R2

  이 경우에는 오퍼랜드 필드가 두 개의 레지스터 번호들만 포함하면 되므로 비트 수가 절약됨.

  ADD R1, X ; R1 ← R1 + M[X]

  이 명령어의 경우에는 명령어 코드 내에 레지스터 번호와 기억장치 주소가 모두 포함되어야 하므로, 주소지정 할 수 있는 기억장치 영역이 상당히 좁아지게 됨.

- 3-주소 명령어

  ADD R1, R2, R3 ; R1 ← R2 + R3

  이 명령어는 세 개의 오퍼랜드들을 포함하지만, 각각은 레지스터 번호이기 때문에 비트 수가 많이 소요되지 않음.

명령어 형식 (5)

• X = (A+B) x (C-D)의 계산을 수행하는 프로그램을 1-주소 명령어, 2-주소 명령어, 3-주소 명령어를 사용하여 구성하면 아래와 같음

* 1-주소 명령어를 사용한 프로그램

```
LOAD   A ; AC ←M[A]
ADD    B  ; AC ← AC + M[B]
STOR   T ; M[T] ←AC
LOAD   C ; AC ← M[C]
SUB    D  ; AC ← AC-M[D]
MUL    T  ; AC ← AC x M[T]
STOR   X ; M[X] ←AC
```

명령어 형식 (6)

* 2-주소 명령어를 사용한 프로그램

```
MOV   R1, A   ; R1 ←M[A]
ADD   R1, B   ; R1 ← R1 + M[B]
MOV  R2, C    ; R2 ← M[C]
SUB   R2, D   ; R2 ← R2-M[D]
MUL   R1, R2  ; R1 ← R1 x R2
MOV   X, R1    ; M[X] ←R1
```

* 3-주소 명령어를 사용한 프로그램

```
ADD   R1, A, B   ; R1 ← M[A] + M[B]
SUB   R2, C, D   ; R2 ← M[C]-M[D]
MUL   X, R1, R2  ; M[X] ← R1 x R2
```

3-주소 명령어를 사용하면 프로그램이 가장 짧아진다는 장점이 있지만 레지스터의 수와 기억장치 용량이 고정된 상태에서 이 명령어 형식을 사용하면 명령어의 비트 수가 늘어나게 되고, 기억장치 용량이 별로 줄어들지 않음.

3 교시

주소지정 방식 (1)

- 주소지정 방식을 설명하기 위해 다음과 같은 표기들을 사용하기로 함.

  EA : 유효 주소(effective address), 즉 데이터가 저장된 기억장치의 실제 주소

  A : 명령어 내의 주소 필드 내용(오퍼랜드 필드가 기억장치 주소인 경우)

  R : 명령어 내의 레지스터 번호(오퍼랜드 필드가 레지스터 번호인 경우)

  (A) : 기억장치 A번지의 내용

  (R) : 레지스터 R의 내용

  명령어가 실행되는 과정에서 주소지정 방식에 따라 유효 주소인 EA가 결정되며,

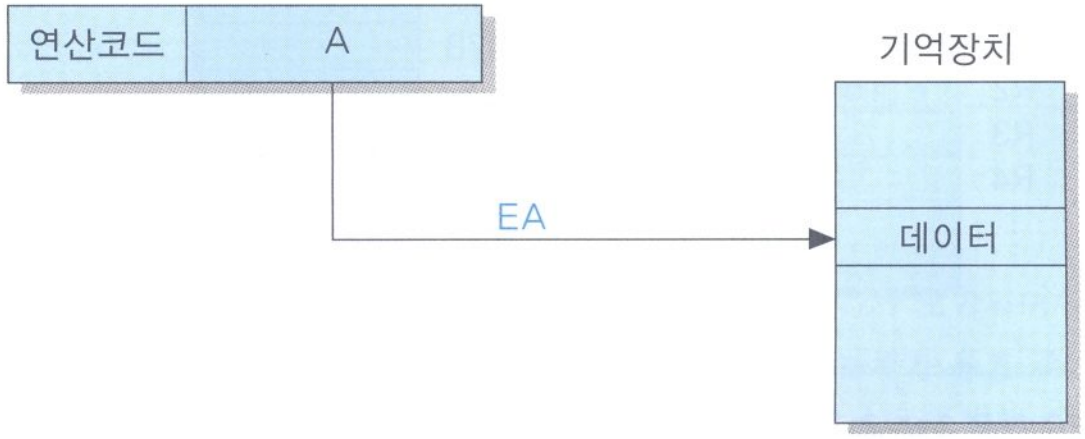
  EA는 실제 데이터를 읽어오기 위한 주소로 사용됨.

  주소지정 방식이 복잡해질수록 EA를 결정하는데 걸리는 시간이 더 길어짐.

주소지정 방식 (2)

(1) 직접 주소지정 방식

- 직접 주소지정 방식(direct addressing mode)은 명령어 내의 오퍼랜드 필드 내용이 데이터의 유효 주소가 되는 간단한 방식임. 즉 EA = A
- 이 방식을 이용하면 데이터 인출을 위하여 한 번의 기억장치 액세스만 필요하며, 유효 주소 결정을 위한 다른 절차나 계산이 필요하지 않음.
- 예로서, LOAD X가 있음.



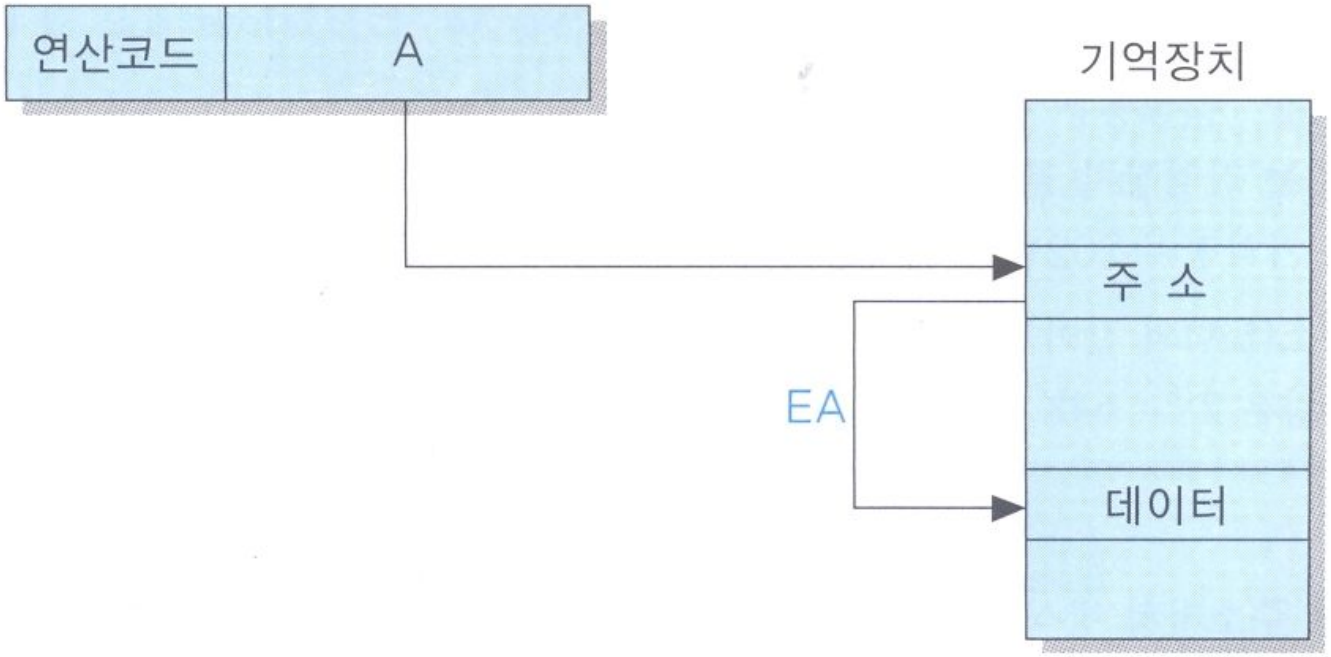
• 직접 주소지정 방식

주소지정 방식 (3)

(2) 간접 주소지정 방식

- 직접 주소지정 방식의 문제점은 주소 필드의 길이가 짧기 때문에 주소를 지정할 수 있는 기억장치 영역이 제한됨.
  이에 대한 해결책으로 명령어의 오퍼랜드 필드에 기억장치 주소가 저장되어 있지만,
  그 주소가 가리키는 기억 장소의 데이터에
  유효 주소(EA)가 저장되어 있도록 할 수 있음.
- 그러면 최대 기억장치 용량이 그 기억 장소에 저장된 전체 비트 수에 의해 결정되므로 더 확장할 수 있음.
  즉 기억장치의 단어 길이가 n비트라면, 최대 2의 n승 개의 주소 공간을 가질 수 있음.
  이 방식을 간접 주소지정 방식이라 하며,
  유효 주소 EA =(A)로 표현함.
- 예로서, LOAD (X)가 있음.

주소지정 방식 (4)



• 간접 주소지정 방식

주소지정 방식 (5)

- 이 방식을 위하여 명령어 형식은 간접 비트(I)가 포함되어야 함. 만약 I비트가 0이면 직접 주소지정 방식을 나타내므로 오퍼랜드 자체가 유효 주소(EA)임. 그러나 만약 I비트가 1이면, 간접 주소지정 방식이므로 오퍼랜드가 지정하는 위치로부터 유효 주소를 인출해야 함.

• 간접 비트가 포함된 명령어 형식

- 이 방식의 단점은 실행 사이클 동안에 두 번의 기억장치 액세스가 필요하다는 점임. 첫 번째 액세스는 주소를 읽어오기 위한 것이고, 두 번째는 그 주소가 지정하는 위치로부터 실제 데이터를 인출하기 위한 것임.
- 간접 주소지정 방식에서는 여러 단계의 간접 지정도 가능함.

  EA = ((..(A)..))

주소지정 방식 (6)

(3) 묵시적 주소지정 방식

- 묵시적 주소지정 방식(implied addressing mode)에서는 명령어 실행에 필요한 데이터의 위치를 지정하지 않아도 묵시적으로 정해져 있음.
  예를 들어서 'SHL' 명령어는 누산기(AC)의 내용을
  좌측으로 쉬프트(shift)하라는 명령어로서,
  오퍼랜드 필드가 없지만 자동적으로 누산기에 대하여 연산이 수행됨.
- 'PUSH R1' 명령어는 레지스터 R1의 내용을 스택에 저장하는데, 이때 스택 포인터(SP)의 내용이 스택의 위치를 지정하는 유효 주소로 사용됨.
- 이 방식의 가장 큰 장점은 명령어 길이가 짧다는 점임.

주소지정 방식 (7)

(4) 즉치 주소지정 방식

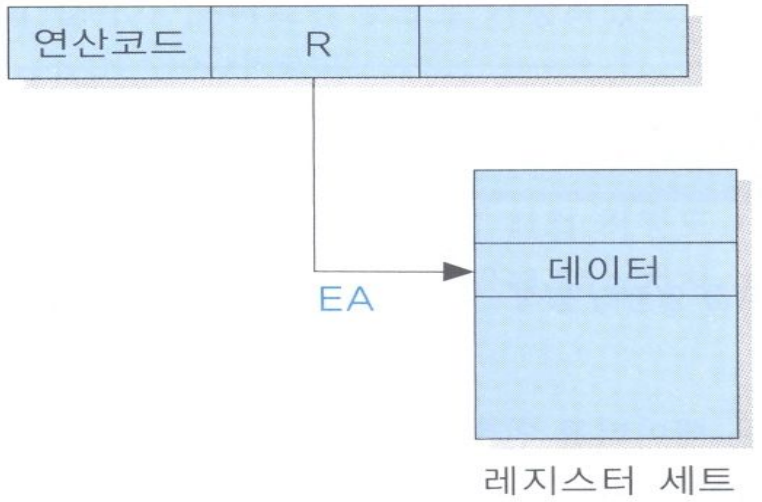
- 즉치 주소지정 방식(immediate addressing mode)에서는 연산에 사용할 데이터가 명령어에 포함되어 있음.
  즉, 오퍼랜드 필드의 내용이 연산에 사용할 실제 데이터임.
  이 방식은 프로그램에서 레지스터들이나 변수의 초기 값을 어떤 상수값으로 세트 하는 데 유용하게 사용됨.
- 즉치 주소지정 방식의 장점은 데이터를 인출하기 위하여 기억장치를 액세스할 필요가 없기 때문에, 실행 사이클이 짧아진다는 점임.
  단점으로는 사용할 수 있는 수의 크기가 오퍼랜드 필드의 수에 의해 제한된다는 점임.

• 즉치 주소지정 방식을 위한 명령어 형식

주소지정 방식 (8)

(5) 레지스터 주소지정 방식

- 레지스터 주소지정 방식(register addressing mode)에서는 연산에 사용할 데이터가 레지스터에 저장되어 있음. 따라서 오퍼랜드 필드의 내용이 레지스터 번호이며, 그 번호가 가리키는 레지스터의 내용이 명령어 실행 과정에서 데이터로 사용됨. 따라서 EA = R이 됨.
- 이 방식의 장점으로는 명령어에서 오퍼랜드 필드의 비트 수가 적어도 되고, 데이터 인출을 위하여 기억장치 액세스가 필요 없으므로 속도가 빠르지만, 단점으로는 데이터가 저장될 수 있는 공간이 CPU 내부 레지스터들로 제한된다는 점임.

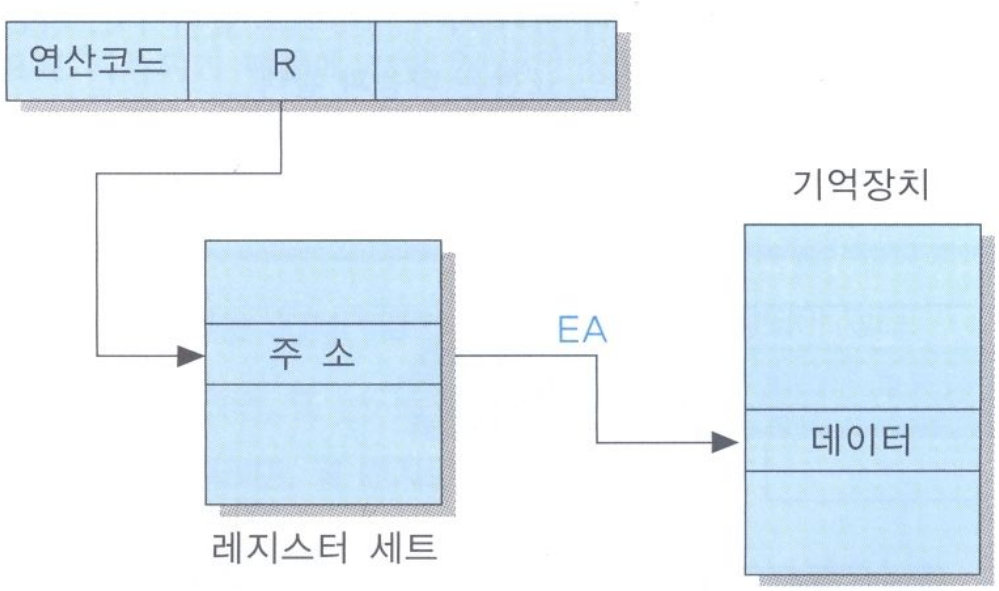


• 레지스터 주소지정 방식

주소지정 방식 (9)

(6) 레지스터 간접 주소지정 방식

• 레지스터 간접 주소지정 방식(register-indirect addressing mode)에서는
오퍼랜드 필드(레지스터 번호)가 가리키는 레지스터의 내용이 유효 주소가 됨.
즉 EA = (R)임.
따라서, 지정된 레지스터의 내용이 기억장치 주소로 사용되어 실제 데이터를 인출하게 됨.

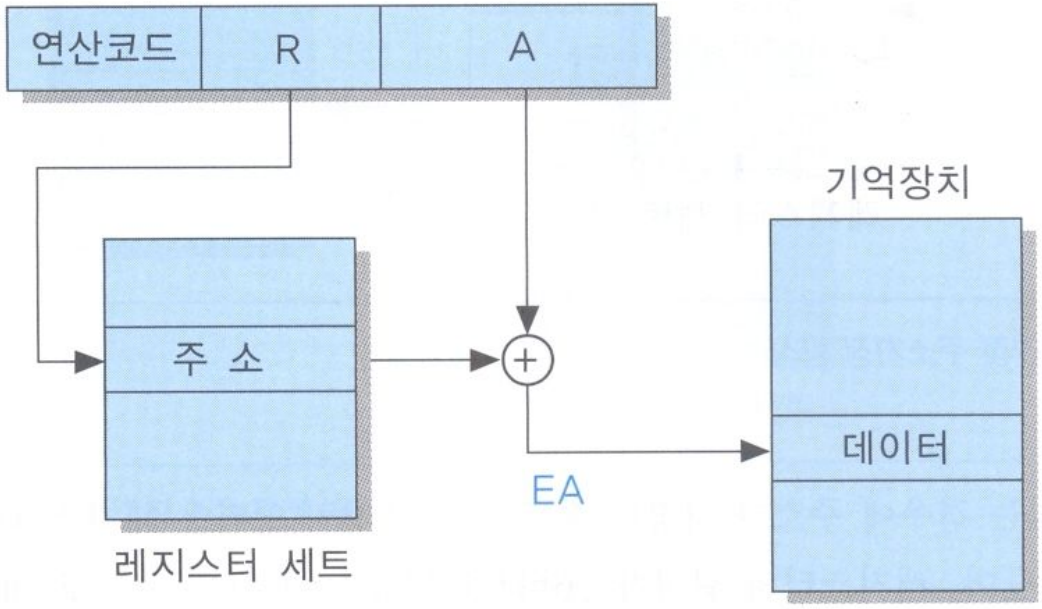


• 레지스터 간접 주소지정 방식

주소지정 방식 (10)

(7) 변위 주소지정 방식

- 직접 주소지정 방식과 레지스터 간접 주소지정 방식을 조합하면 변위 주소지정 방식 (displacement addressing mode)이라는 강력한 주소지정 방식을 만들 수 있음.
- 이 방식들은 사용되는 레지스터에 따라 여러 가지 명칭으로 불리지만, 기본 메커니즘은 동일하며, EA = A +(R)의 형태임.



• 변위 주소지정 방식

주소지정 방식 (11)

- 사용되는 레지스터에 따라 여러 종류의 변위 주소지정 방식들이 정의될 수 있으며, 가장 널리 사용되고 있는 대표적인 것들로는 다음과 같은 세 가지가 있음.

(1) 상대 주소지정 방식(relative addressing mode)

- 이 방식에서는 레지스터로서 프로그램 카운터(PC)가 사용되므로 EA = A + (PC)임. 즉 PC의 내용과 변위 A를 더하여 유효 주소를 결정함.
  그런데 현재 PC의 내용은 다음에 실행할 명령어의 주소이므로,
  변위는 그 명령어의 위치를 기준으로 한 상대적인 값이 됨.
  이 방식은 주로 분기 명령어에서 사용되는데, 변위가 양수인 경우에는 앞 방향으로 분기하고, 음수인 경우에는 뒷 방향으로 분기함.
  이 방식을 사용하면 변위의 범위가 오퍼랜드 필드의 길이에 의해 제한되지만,
  전체 기억장치 주소가 명령어에 포함되어야 하는 일반적인 분기 명령어보다
  적은 수의 비트만 있으면 된다는 장점이 있음.

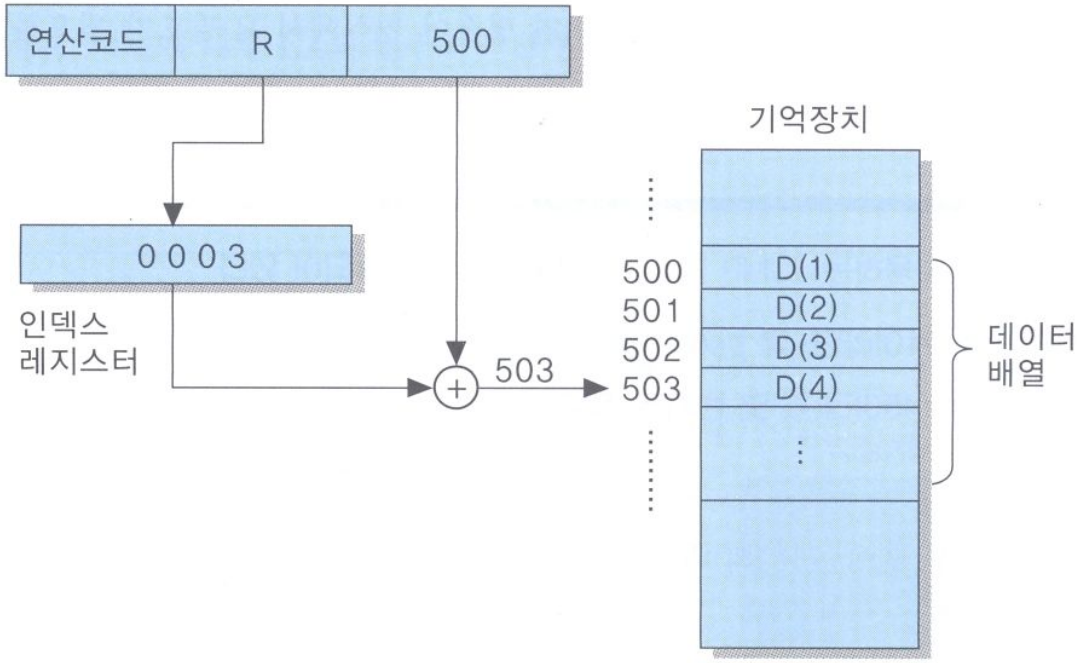
주소지정 방식 (12)

(2) 인덱스 주소지정 방식(indexed addressing mode)

- 이 방식에서는 인덱스 레지스터의 내용과 변위 A를 더하여 유효 주소를 결정함.
  인덱스 레지스터는 인덱스(index) 값을 저장하는 특수 레지스터인데, 이 레지스터를 IX라고 하면, EA = A + (IX)가 됨.
- 일반적으로 주소 A는 기억장치에 저장된 데이터 배열(data array)의 시작 주소를 가리키고, 인덱스 레지스터의 내용은 그 배열의 시작 주소로부터 각 데이터까지의 거리를 나타냄.
  따라서 인덱스 레지스터의 내용을 적절히 변경하면, 프로그램 루프(program loop) 내에서 동일한 명령어를 사용하여 배열 내의 서로 다른 데이터들을 액세스 하는 것이 가능해짐.

주소지정 방식 (13)

- 아래 그림에서 데이터 배열이 기억장치의 500번지부터 저장되어 있고, 그 시작 주소를 가리키기 위하여 명령어의 주소 필드에 500이 포함되어 있음.
- 그리고 배열의 시작 주소로부터 각 데이터까지의 거리를 나타내는 인덱스 값은 인덱스 레지스터에 저장되어 있음. 그림은 인덱스 레지스터에 3이 저장되어 있어서 네 번째 데이터가 액세스되는 경우를 보여주고 있음.



• 인덱스 주소지정 방식의 예

주소지정 방식 (14)

(3) 베이스-레지스터 주소지정 방식(base-register addressing mode)

- 이 방식에서는 베이스 레지스터의 내용과 변위 A를 더하여 유효 주소를 결정함. 즉, 베이스 레지스터를 BR이라고 하면, EA = A + (BR)이 됨.
- 이 방식은 인덱스 주소지정 방식과 유사한데, 두 방식의 차이점은 레지스터가 사용되는 방법에 있음.
- 인덱스 레지스터에는 명령어에 포함된 주소 A를 기준으로 한 인덱스 값이 저장되는 반면에, 베이스 레지스터에는 기준이 되는 주소가 저장됨.
- 용도 상의 차이를 보면, 인덱스 주소지정 방식은 데이터 배열의 액세스에 주로 사용되지만, 베이스-레지스터 주소지정 방식은 기억장치 내의 프로그램의 시작 위치를 지정하는 데 사용됨.

셀프 테스트

• CALL X 명령어에서 복귀할 주소를 어디에 저장해야 하는가?

* 스택

해설) CALL X라는 명령어는 X번지부터 시작되는 서브루틴을 실행하라는 명령어이며 이 서브루틴이 끝나고 되돌아오기 위해 복귀할 주소를 서브루틴 시작하기 전에 스택에 저장함.

• 명령어의 연산 코드 비트 수가 5이면 CPU가 수행할 수 있는 명령어의 수는 몇 개인가?

* 32

해설) 연산코드 비트 수가 5이면 명령어의 종류는 2의 5승, 즉 32개까지 될 수 있음.

• PUSH R1의 명령어에서 사용되는 주소지정 방식은 무엇인가?

* 묵시적 주소지정 방식

해설) 묵시적 주소지정방식은 데이터의 위치가 묵시적으로 정해져 있는 방식인데 PUSH R1 명령어에서는 데이터의 위치가 SP가 되는 것임.

요점 정리

- CPU가 수행할 수 있는 연산(operation)의 종류에는 데이터 전송, 산술 연산, 논리 연산, 입출력 I/O, 프로그램 제어 등이 있음.
- 각 명령어는 일련의 비트들에 의해 표현되며, 이 비트들은 편의상 여러 개의 필드(field)들로 나눌 수 있는데, 필드들의 수와 배치 방식 및 각 필드의 비트 수를 명령어 형식이라고 함.
- 명령어가 한 개, 두 개, 세 개의 오퍼랜드를 포함하는 것을 각각 1-주소 명령어 (one-address instruction), 2-주소 명령어(one-address instruction), 3-주소 명령어(one-address instruction) 라고 함.
- 주소지정 방식에는 직접 주소지정 방식, 간접 주소지정 방식, 묵시적 주소지정 방식, 즉치 주소지정 방식, 레지스터 주소지정 방식, 레지스터 간접 주소지정 방식, 변위 주소지정 방식 등이 있음.
- 사용되는 레지스터에 따라 여러 종류의 변위 주소지정 방식들이 정의될 수 있으며, 여기에는 상대 주소지정 방식, 인덱스 주소지정 방식, 베이스-레지스터 주소지정 방식 등이 있음.