주소지정(Addressing) 방식

05 주소 지정 방식

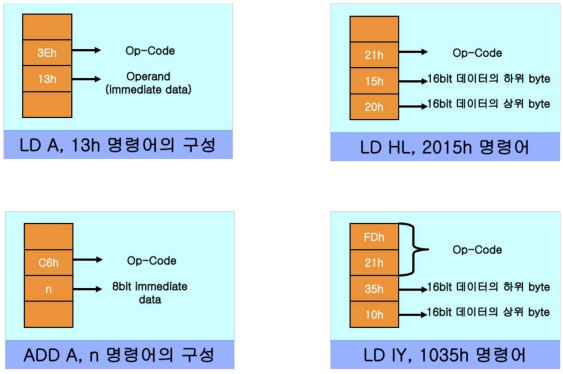
1 즉시 주소 지정(immediate addressing mode, 즉치 주소 지정)

- 오퍼랜드를 지정하는 가장 간단한 방법
  - 명령어 자체에 오퍼랜드를 포함
  - 오퍼랜드가 포함되어 명령어가 인출될 때 오퍼랜드도 자동으로 인출
  - 즉시(**즉치**) 오퍼랜드 : 즉시 사용 가능
- 레지스터 R1에 상수 4를 저장하는 즉시 주소 지정 명령어의 예

MOVE	R1	4

그림 4-13 즉시 주소 지정

명령어 안에 숫자 값이 들어 있어 따로 메모리를 참조할 필요가 없음



- 장점 : 오퍼랜드를 인출을 위한 메모리 참조가 필요 없음
- 단점 : 상수만 가능, 상수 값의 크기가 필드 크기로 제한
- 작은 값의 정수를 지정하는 데 많이 사용

2 직접 주소 지정(direct addressing mode)

- 메모리에 위치한 오퍼랜드의 전체 주소 지정
- 직접 주소 지정도 즉시 주소 지정처럼 사용 제한
  - 명령어는 항상 정확히 동일한 메모리 위치 액세스
  - 값이 변할 수는 있지만 위치는 변할 수 없음
  - 컴파일할 때 알려진 주소의 전역 변수에 액세스하는 데만 사용 가능

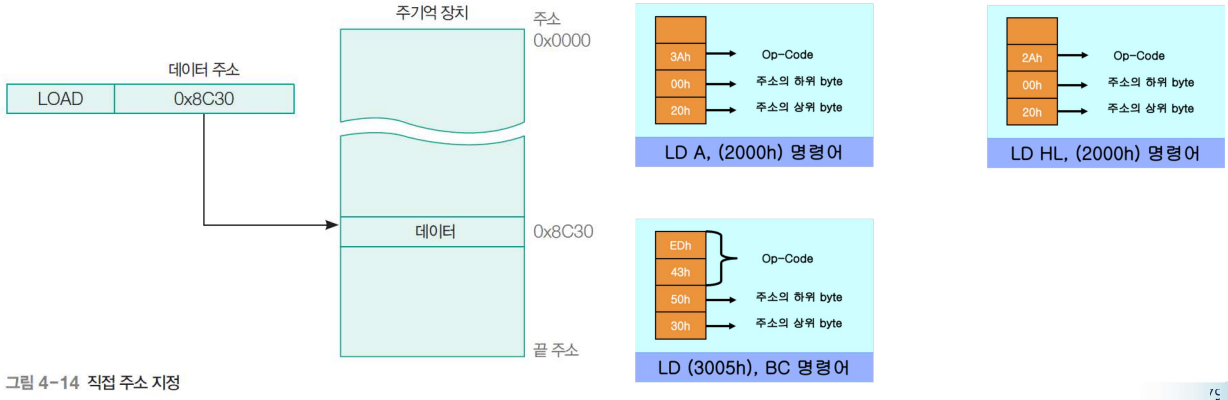


그림 4-14 직접 주소 지정

3 레지스터 주소 지정(register addressing mode)

- 직접 주소 지정과 개념은 같고 그 위치가 메모리 대신 레지스터 ← 레지스터 안에 즉치 값이 있음
- 가장 일반적인 주소 지정 방식:
  - 레지스터는 액세스가 빠르고 주소가 짧기 때문
  - 대부분의 컴파일러는 루프 인덱스처럼 가장 자주 액세스할 변수를 레지스터에 넣기 위해 많은 노력을 기울임
- 많은 프로세서에서 사용
- RISC 등에서는 LOAD, STORE 명령을 제외하고 대부분의 명령어에서 레지스터 주소 지정 방식만 사용
- LOAD나 STORE 명령어
  - 한 오퍼랜드는 레지스터고,
    다른 한 오퍼랜드는 메모리 주소

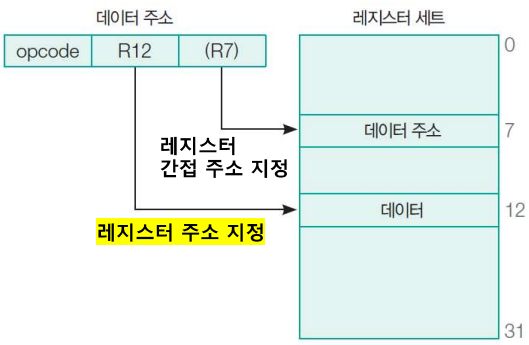


그림 4-15 레지스터 주소 지정

05 주소 지정 방식

4 레지스터 간접 주소 지정(register indirect addressing mode)

- 직접 주소를 명령어에는 포함하지 않음
  - 메모리의 주소는 레지스터에 저장: 포인터(pointer)
  - 레지스터 간접 주소 지정의 가장 큰 장점 : 명령어에 전체 메모리 주소가 없어도 메모리 참조가능

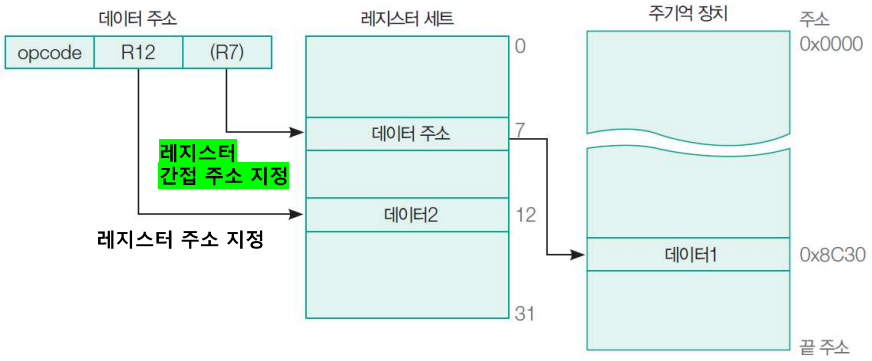


그림 4-16 레지스터 간접 주소 지정

05 주소 지정 방식

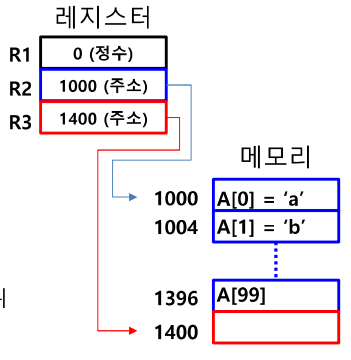
❖ **레지스터 R1에 있는 요소의 합계 계산 예**

요소가 100개인 1차원 long(4바이트) 배열의 요소를 단계별로 설명하는 루프를 생각해 보자.
루프 외부에서는 R2 같은 다른 레지스터를 배열의 첫 번째 요소를 가리키도록 설정할 수 있으며
다른 레지스터, 예를 들어 R3은 배열을 벗어나는 첫 번째 주소를 가리키도록 설정할 수 있다.
배열이 4바이트(32비트 정수)인 정수 100개가 있는 경우 배열이 A에서 시작하면 배열을 벗어나는 첫 번째 주소는 A+400이 된다. 이 계산을 수행하는 일반적인 어셈블리 코드는 다음과 같다.

```
        1  MOVE R1, 0          ; 계산 결과가 저장될 R1에 초깃값 0 저장
        2  MOVE R2, A          ; R2는 배열 A의 주소
        3  MOVE R3, A+400      ; R3은 배열 A를 벗어나는 첫 번째 주소
LOOP:   4  ADD R1, (R2)        ; R2를 이용하여 간접 주소를 지정하고 피연산자를 가져옴
        5  ADD R2, 4           ; R2 레지스터를 4만큼 증가시킴(4바이트), 바이트 단위 주소 지정
        6  CMP R2, R3          ; R2와 R3를 비교, 즉 끝에 도달였는가를 판단하기 위함
        7  BLT LOOP            ; R2<R3이면 LOOP로 가서 반복함
```

```
long A[100];
R1 = 0;
R2 = A;
R3 = A+400;

while(R2<R3) {
  R1 += *R2;
  R2 += 4;
}
```



- 특이한 점 : 루프 자체에 메모리 주소가 포함되지 않음 → 레지스터 액세스로 속도가 빠름
- 네 번째 명령어 : 레지스터 주소 지정과 레지스터 간접 주소 지정
- 다섯 번째 명령어 : 레지스터 주소 지정과 즉시 주소 지정을
- 여섯 번째 명령어 : 둘 다 레지스터 주소 지정

- BLT(Branch Less Than) : 메모리 주소를 사용 가능하지만, BLT 명령어 자체에 상대적인 8비트 변위로 분기할 주소를 지정할 때가 많음
- 메모리 주소의 사용을 완전히 피함으로써 짧고 빠른 루프 가능

05 주소 지정 방식

5 변위 주소 지정(displacement addressing mode)

- 특정 레지스터에 저장된 주소에 변위(offset: 오프셋)을 더해 실제 오퍼랜드가 저장된 메모리 위치 지정
- 특정 레지스터가 무엇인지에 따라 여러 주소 지정 방식 가능
- 예 : 인덱스 주소 지정 방식은 레지스터를 인덱스로 사용
  상대 주소 지정 방식에서는 PC(프로그램카운터)가 특정 레지스터로 지정

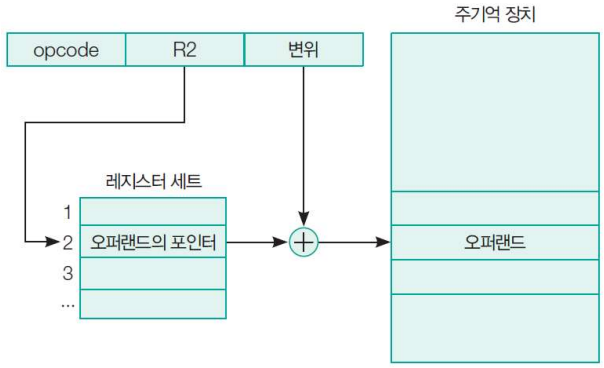


그림 4-17 변위 주소 지정

05 주소 지정 방식 -- 변위 주소 지정(displacement addressing mode)

❑ 인덱스 주소 지정(indexed addressing mode)

- 레지스터(명시적 또는 암시적)에 일정한 변위를 더해 메모리 주소 참조
- 특정 레지스터가 무엇인지에 따라 여러 주소 지정 방식 가능
- 예 : 인덱스 주소 지정 방식은 인덱스 레지스터가 되고, 상대 주소 지정 방식에서는 PC가 특정 레지스터로 지정

```
            MOVE R1, 0          ; R1은 합이 저장될 장소이며, 초깃값으로 0을 설정
            MOVE R2, 0          ; R2는 배열 A의 인덱스 i가 저장될 장소이며, 4씩 증가됨
            MOVE R3, 400        ; R3 400이 될 때까지 4씩 증가함
LOOP:       ADD R1, A(R2)       ; R1 = R1 + A[i]
            ADD R2, 4           ; i = i + 4(워드 크기 = 4bytes)
            CMP R2, R3          ; 100개 모두 계산되었는지 비교
            BLT LOOP
```

```
long A[100];
R1 = 0;
R2 = 0;
R3 = 100;

while(R2<R3) {
  R1 += A[R2];
  R2 += 1;
}
```

- 프로그램의 알고리즘 : 단순하며 3개의 레지스터 필요
  ❶ R1 : A의 누적 합계가 저장된다.
  ❷ R2 : 인덱스 레지스터로 배열의 i를 저장한다.
  ❸ R3 : 상수 400
- 명령어 루프에 4개 실행
  - 소스 값의 계산은 인덱스 주소지정 사용
  - 배열 A의 인덱스가 저장된 인덱스 레지스터 R2와 상수(0~400)가 더해져 메모리를 참조(A(R2))하는 데 사용

05 주소 지정 방식

```
MOV R1, A(R2)
```

- R1이 목적지인 레지스터 주소 지정
- 소스는 R2가 인덱스 레지스터이고, A가 변위인 인덱스 주소 지정

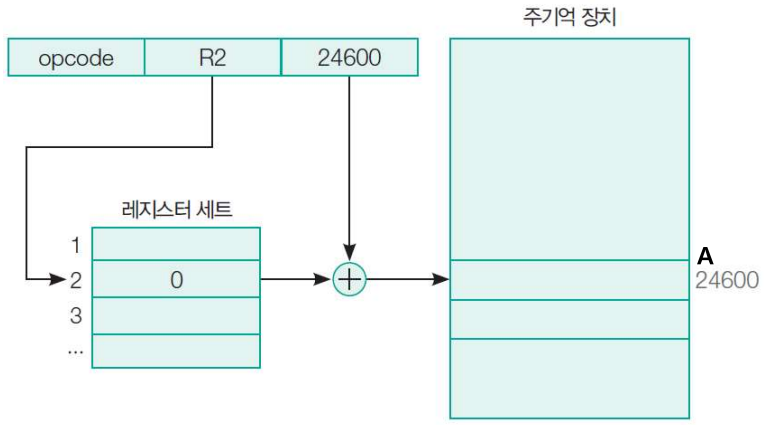


그림 4-18 인덱스 주소 지정

05 주소 지정 방식

❑ 상대 주소 지정(relative addressing mode)

- PC 레지스터 사용
- 현재 프로그램 코드가 실행되고 있는 위치에서 앞 또는 뒤로 일정한 변위만큼 떨어진 곳의 데이터 지정

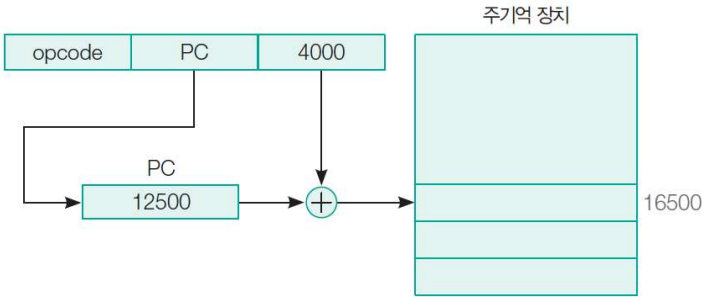
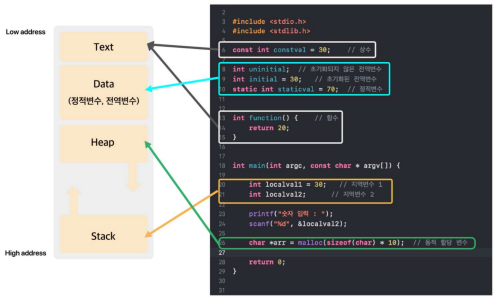


그림 4-19 상대 주소 지정

05 주소 지정 방식

❑ 베이스 주소 지정(base addressing mode)

- 인텔 프로세서에는 세그먼트 레지스터가 6개 있음
- 이 중 하나를 베이스 레지스터로 하고 이 레지스터에 변위를 더해 실제 오퍼랜드가 있는 위치를 찾는 방식
- SS(stack segment): 스택 데이터가 저장되어 있는 스택 위치에 대한 포인터
- CS(code segment): 프로그램 코드가 저장되어 있는 시작 위치에 대한 포인터
- DS(data segment): 데이터 영역에 대한 시작 포인터
- ES, FS, GS: 엑스트라 세그먼트(extra segment)에 대한 포인터
  - 엑스트라 세그먼트는 데이터 세그먼트의 확장 영역



05 주소 지정 방식

- 이 레지스터 중 하나를 베이스로 사용하여 실제 오퍼랜드 위치 지정
  - 데이터 세그먼트를 베이스로 사용: 오프셋 200만큼 떨어진 주소 25600에서 오퍼랜드 위치함

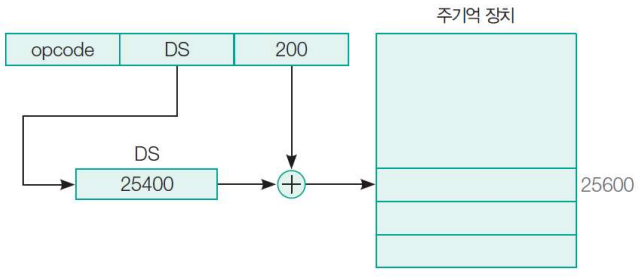


그림 4-20 베이스 주소 지정

05 주소 지정 방식

6 간접 주소 지정(indirect addressing mode)

- 메모리 참조가 두 번 이상 일어나는 경우
- 데이터를 가져오는 데 많은 시간 소요
- 프로세서와 주기억 장치간의 속도 차가 많은 현재의 프로세서의 경우
  - 오퍼랜드를 인출하는 데 오래 걸리므로 전체 프로그램의 수행 시간은 길어짐
  - 현재는 간접 주소 지정을 지원하는 프로세서는 거의 없음

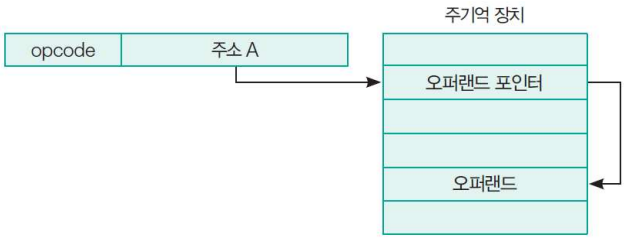
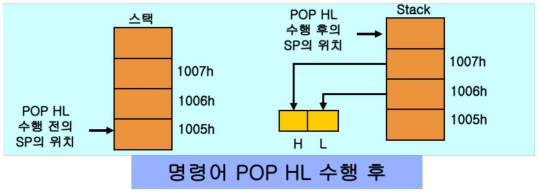


그림 4-21 간접 주소 지정

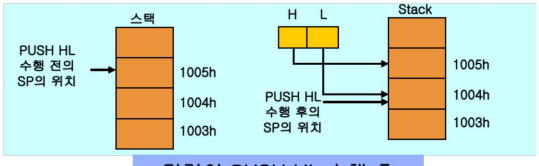
05 주소 지정 방식

7 묵시적 주소 지정(implied addressing mode)

- 오퍼랜드의 소스나 목적지를 명시하지 않음
- 암묵적으로 그 위치를 알 수 있는 주소 지정 방식
- 예를 들어 서브루틴에서 호출한 프로그램으로 복귀할 때 사용하는 RET 명령
  - 명령어 뒤에 목적지 주소가 없지만 어디로 복귀할지 자동으로 알 수 있음
- PUSH, POP 등 스택 관련 명령어
  - 스택이라는 목적지나 소스가 생략
  - PUSH R1 : 레지스터 R1의 값을 스택에 저장
  - POP : 스택의 TOP에 있는 값을 AC로 인출
- 누산기를 소스나 목적지로 사용하는 경우도 생략 가능



명령어 POP HL 수행 후



명령어 PUSH HL 수행 후

05 주소 지정 방식

```
ModR/M value        C8  = 11001000
                    ----------------
                    Mod = 11
                    RM  =      000
/digit (Opcode) or REG =   001
```

8 코어 i7의 주소 지정 방식

- 즉시, 직접, 레지스터, 레지스터 간접, 인덱싱 및 배열 요소 주소 지정을 위한 특수 모드가 있음

표 4-3 코어 i7 32비트 주소 지정 방식, M[x]는 주소 x의 메모리 워드

R/M \ MOD	00	01	10	11
000	M[EAX]	M[EAX+OFFSET8]	M[EAX+OFFSET32]	EAX 또는 AL
001	M[ECX]	M[ECX+OFFSET8]	M[ECX+OFFSET32]	ECX 또는 CL
010	M[EDX]	M[EDX+OFFSET8]	M[EDX+OFFSET32]	EDX 또는 DL
011	M[EBX]	M[EBX+OFFSET8]	M[EBX+OFFSET32]	EBX 또는 BL
100	SIB	SIB with OFFSET8	SIB with OFFSET32	ESP 또는 AH
101	Direct	M[EBP+OFFSET8]	M[EBP+OFFSET32]	EBX 또는 CH
110	M[ESI]	M[ESI+OFFSET8]	M[ESI+OFFSET32]	ESI 또는 DH
111	M[EDI]	M[EDI+OFFSET8]	M[EDI+OFFSET32]	EDI 또는 BH

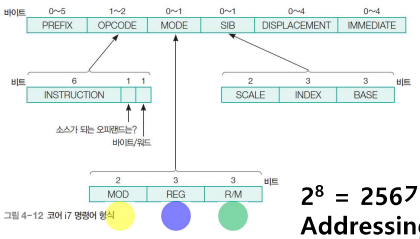


그림 4-12 코어 i7 명령어 형식

2^8 = 256가지
Addressing Form 있음

Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r)	Second operand registers if an instruction requires one	AL AX EAX MM0 XMM0	CL CX ECX MM1 XMM1	DL DX EDX MM2 XMM2	BL BX EBX MM3 XMM3	AH SP ESP MM4 XMM4	CH BP[1] EBP MM5 XMM5	DH SI ESI MM6 XMM6	BH DI EDI MM7 XMM7
(In decimal) /digit (Opcode) (In binary) REG	8 Reg/Opcode (Decimal/Binary)	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)						
[BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] disp16[2] [BX]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[BX+SI]+disp8[3] [BX+DI]+disp8 [BP+SI]+disp8 [BP+DI]+disp8 [SI]+disp8 [DI]+disp8 [BP]+disp8 [BX]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A	60 61 62	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[BX+SI]+disp16 [BX+DI]+disp16 [BP+SI]+disp16 [BP+DI]+disp16 [SI]+disp16 [DI]+disp16 [BP]+disp16 [BX]+disp16	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM1/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AHMM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

24 Addressing mode
8 R/M
4 Mod
256 ModR/M possible values
8 registers

05 주소 지정 방식

9 실제 프로세서에서 주소 지정 방식

- Core i7, ARM 및 AVR에서 사용되는 주소 지정 방식

표 4-4 주요 프로세서의 주소 지정 방식 비교

주소 지정 방식	X86	ARM	AVR
즉시 주소	O	O	O
직접 주소	O		O
레지스터 주소	O	O	O
레지스터 간접 주소	O	O	O
인덱스 주소	O	O	
베이스-인덱스 주소		O	