8051 어셈블리어

학습 목표

- 프로그래밍 언어의 종류를 분류하고 특성을 이해한다.
- 8051에서 사용되는 주소지정방식을 이해한다.
- 8051 어셈블리 명령어의 사용법을 이해한다.
- 8051 어셈블리 언어에서 사용되는 지시어를 이해한다.

1. 프로그래밍 언어

1.1 기계어

What is the program language?

- Program : CPU가 순서적으로 처리해야 할 일을 만드는 것.
- 명령어(Instruction) : CPU가 처리해야 할 일.

Machine Language

CPU가 이해할 수 있는 언어.

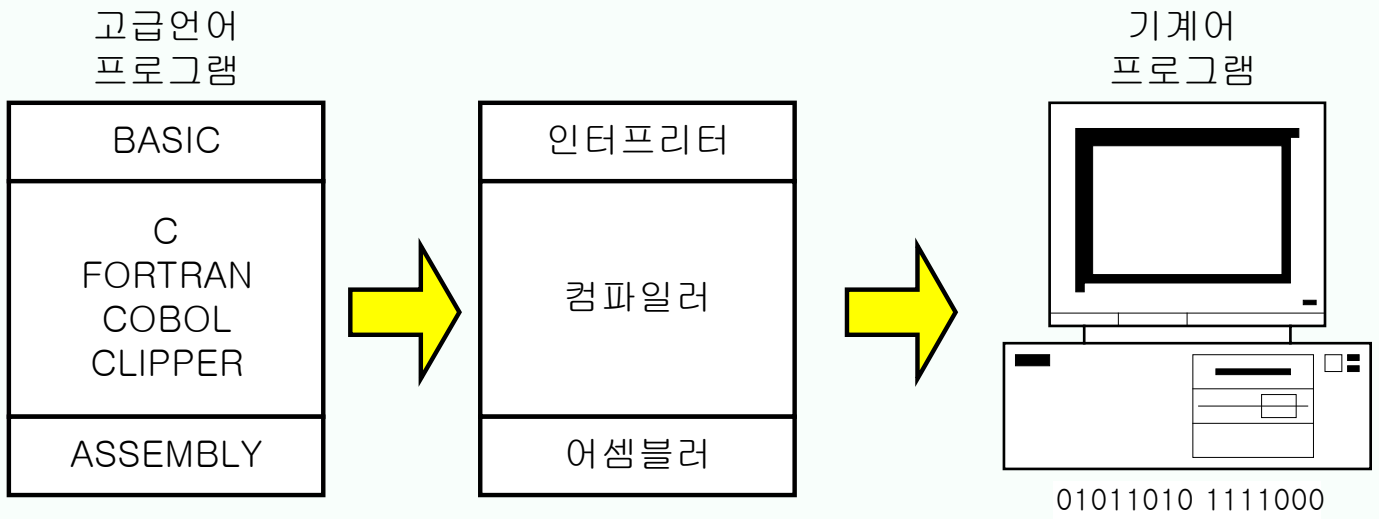
Mnemonic Code

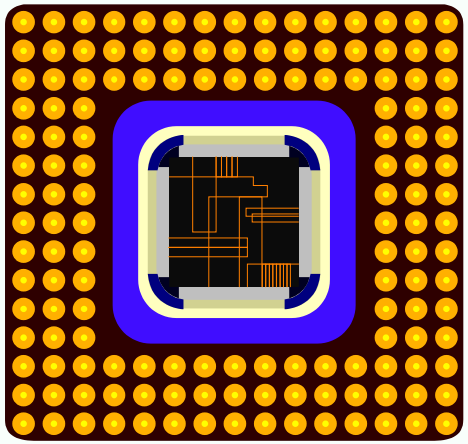
사람이 이해하기 쉽도록 기호 또는 문자를 압축해서 만든 코드.

프로그래밍 언어의 종류

- 고급 언어 (High level language) : FORTRAN, PASCAL, COBOL, C 언어 등
- 저급 언어 (Low level language) : 어셈블리어
- 기계어 (Machine language) : 기계 고유의 언어

❑ 고급 언어의 컴파일 과정





8051 CPU

이해한다

Machine Language

이해할 수 없다

Assembly Language

1.2 어셈블리어

어셈블리어

기계어의 비트 형식을 니모닉 코드(mnemonic code)로 나타낸 것

니모닉 코드(mnemonic code)

기계어의 비트 형식이 나타내는 의미를 심벌(symbol)로 표현한 것으로 프로그램을 이해하거나 작성하기가 쉽다.

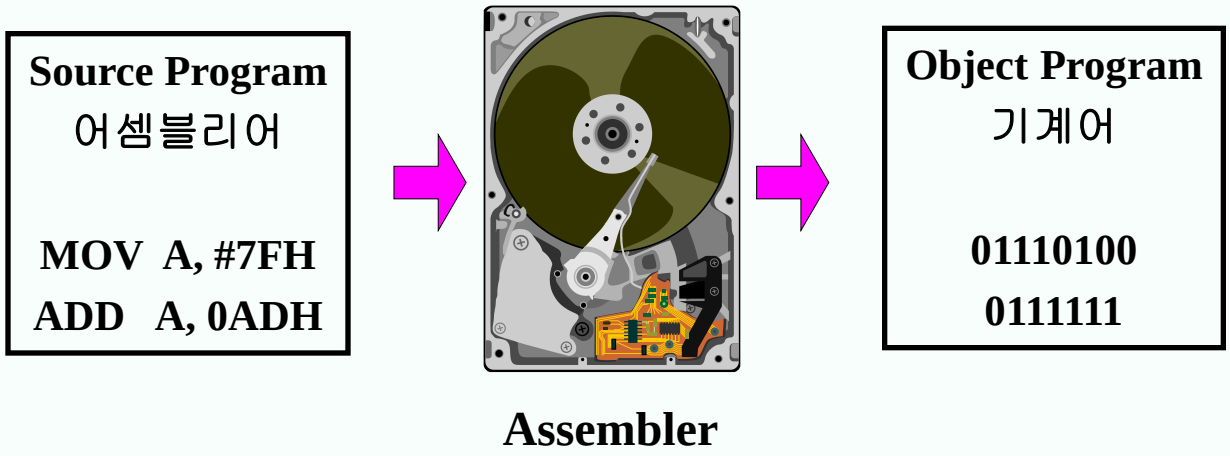
```
MOV   A, #03H
```

어셈블리어로 프로그래밍을 하는 이유

- 컴퓨터 하드웨어의 구성 요소들을 직접 액세스하려고 할 때
- 컴파일러를 설계하거나 시스템 프로그램을 작성하려고 할 때
- 빠른 수행이 필요한 프로그램을 작성하려고 할 때
- 기억 장소를 적게 차지하거나 입출력 장치를 보다 효율적으로 사용하려는 경우

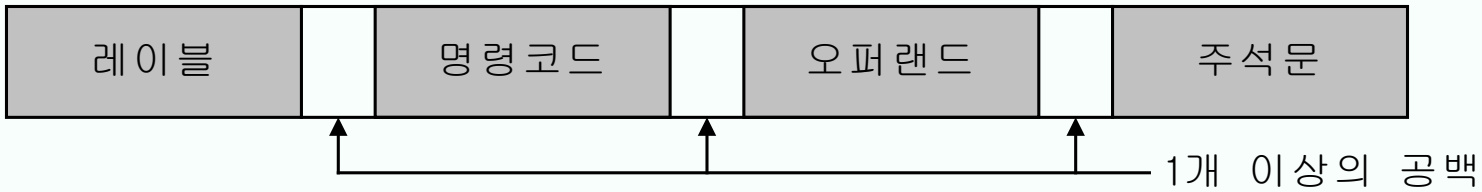
Assembler

- Assembler : 어셈블리어를 기계어로 변환하는 프로그램
- Source Program : 어셈블리어 프로그램
- Object Program : 변환된 기계어 프로그램



2. 8051 어셈블리어

2.1 어셈블리어의 형식



레이블(label) 부

- 레이블은 기호주소(**symbol address**)로서 점프 혹은 서브루틴콜 명령 등에서 사용한다.
- **8**문자 이내의 영문자 또는 숫자를 사용하고, 레이블 중에 공백이 없어야 하며, 최초의 문자는 반드시 영문자이어야 한다.
- 예약어(명령어 또는 어셈블리 지시어)를 레이블로 사용할 수 없다.
- 레이블 바로 뒤에는 콜론(:)을 둔다. 레이블과 콜론 사이에는 공간을 두지 않는다.
- 레이블은 소문자와 대문자의 구분이 있다.
- 하나의 프로그램 내에 같은 레이블이 두 군데 이상 있어서는 안된다.

명령코드(Operation) 부

- 명령의 니모믹(mnemonic) 및 의사 명령어 등을 쓴다.
- 니모닉은 어셈블리 언어에 예약되어 있는 MOV, ADD 등과 같은 명령어이며, 의사 명령어는 프로그램 실행과 관계없이 어셈블러에게 정보를 제공해 주는 지시어이다.

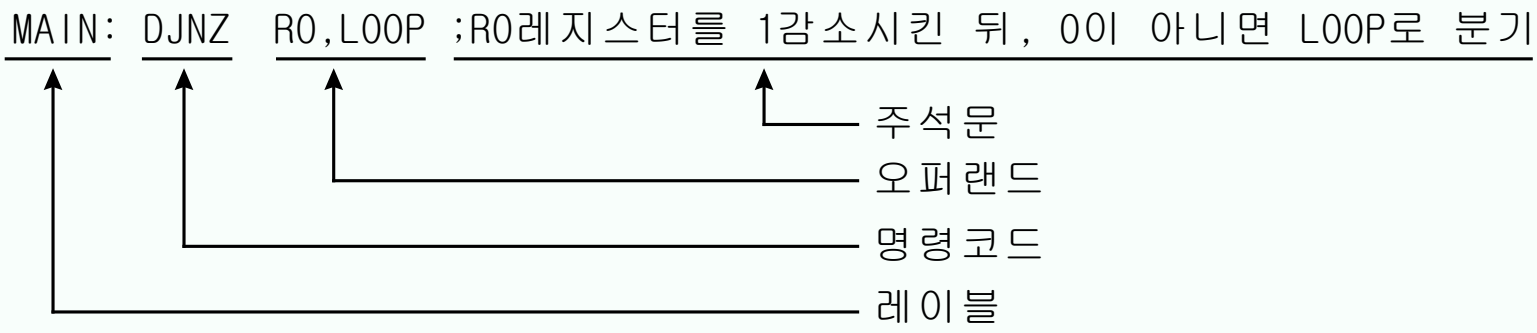
오퍼랜드(Operand) 부

- 오퍼랜드는 명령의 대상이 되는 데이터, 또는 그것이 들어있는 주소 또는 기준주소로부터 떨어져 있는 정도(offset)를 의미한다. 레지스터 이름, 정수, 레이블, 연산자, 주소 등이 해당된다.
- 오퍼랜드부가 필요 없으면 생략 가능하다
- 데이터가 숫자인 경우 진수 표시 방법

진법	표시	예
10진수	<숫자> 또는 <숫자>D	728, 728D
16진수	<숫자>H	7FH, 0E1H
8진수	<숫자>O	427O
2진수	<숫자>B	10010111B
문자상수	'<문자>'	'A'
문자열상수	"<문자>"	"PROCESSOR"

주석(Command) 부

- 프로그램 설명(주석)에 사용되며 생략 가능하다.
- 주석을 붙일 때에는 반드시 세미콜론(;)으로 시작하여야 한다.
- 어셈블러는 주석문을 무시하기 때문에 아무데나 사용하여도 된다.
- 프로그래머 자신이나 다른 사람의 이해를 돕기 위하여 사용하는 것에 불과하므로 어셈블러는 이를 완전히 무시하고 어셈블한다.



어셈블리어의 작성 예

2.2 어셈블러 지시어

"어셈블러 지시어"란 프로그램 실행과는 관계가 없고 단지 어셈블러에게 정보만 제공해 주는 명령어이다. 어셈블러 지시어는 의사 명령어(Pseudo Instruction)라고도 한다.

ORG (origin)

- 프로그램과 데이터의 시작 번지를 설정할 때 사용한다.

```
[Example]    ORG    8000H
             MOV    SP,#50H
                    ……
```

※ 첫번째 기계어는 8000H 번지에 저장되고 그 이하의 명령어들은 이어서 저장된다.

EQU (equate)

- 레이블에 어떤 값을 할당하는데 사용한다.

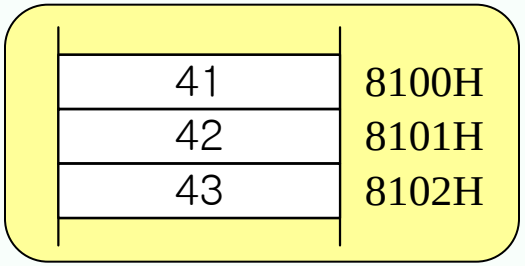
```
[Example]    LCDCLR    EQU    7030H
```

※ 이 문장 이후에 나타나는 LCDCLR은 모두 7030H라는 수치 데이터로 치환된 후에 어셈블된다.

DB (Define Byte)

- 메모리에 숫자 또는 문자 데이터를 1 바이트 단위로 저장하는데 사용

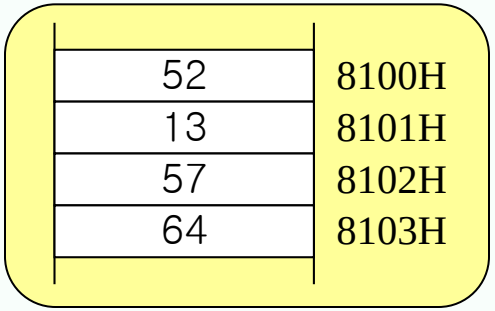
```
[Example]   ORG   8100H
            DB    'ABC'
            ……
```



DW (Define Word)

- 메모리에 숫자 또는 문자 데이터를 2바이트(워드) 단위로 저장하는데 사용

```
[Example]   ORG   8100H
            DB    5213H, 5764H
            ……
```



DS (Define Storage)

- 메모리를 바이트 단위로 확보해 두는 의사 명령어이다. 즉, DS 다음에 오는 바이트의 수만큼 기억공간을 비워 놓는다.

  [Example] BUFFER DS 5

  ※ BUFFER라는 기호화된 주소부터 시작하여 5바이트의 기억공간이 미리 확보된다.

BIT

- 비트 주소를 지정하기 위하여 사용되는 의사 명령어이다. 즉, 8051의 내부 데이터 메모리(RAM)의 20H~2FH 번지까지의 비트 어드레싱 영역에 위치한 특정 비트를 정의하기 위하여 사용된다.

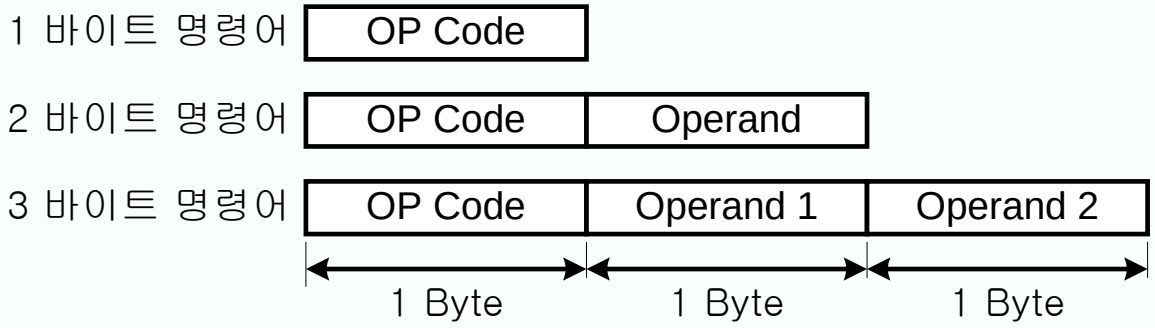
  [Example] COUNTER BIT 20H

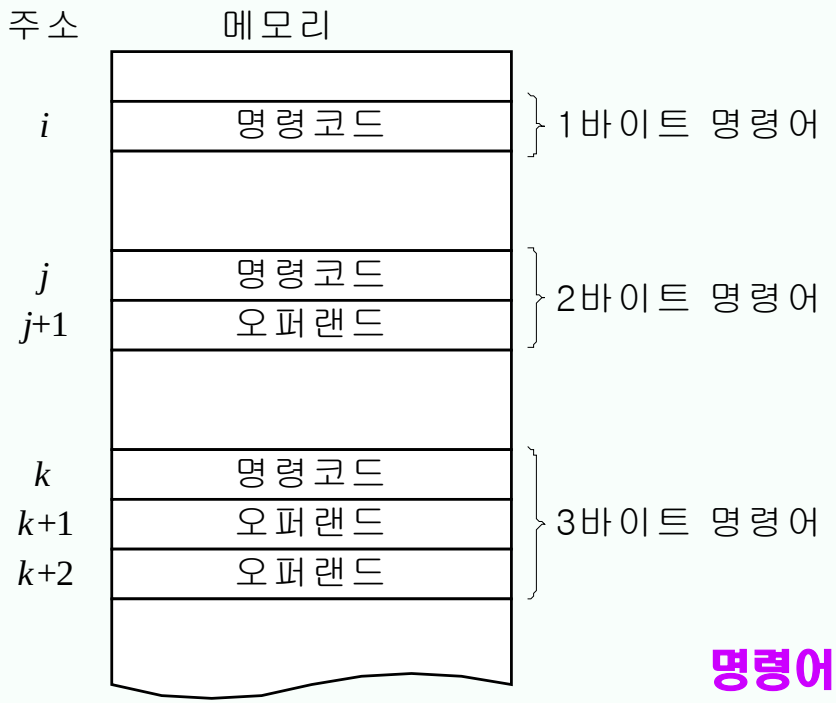
  ※ 비트 어드레싱 영역의 비트 주소 20H, 즉 24H 번지의 첫번째 비트(24.0H)를 COUNTER로 지정한다.

END

- 원시 프로그램의 마지막을 알려주는 의사 명령어이다. END문 이후의 어셈블리어 프로그램은 기계어로 변환되지 않는다.

2.3 명령어 형식





명령어가 메모리에 저장되어 있는 형태

1바이트 명령어

명령어	명령코드	동작
NOP	00H	No operation
INC A	04H	(A) ← (A)+1

2바이트 명령어

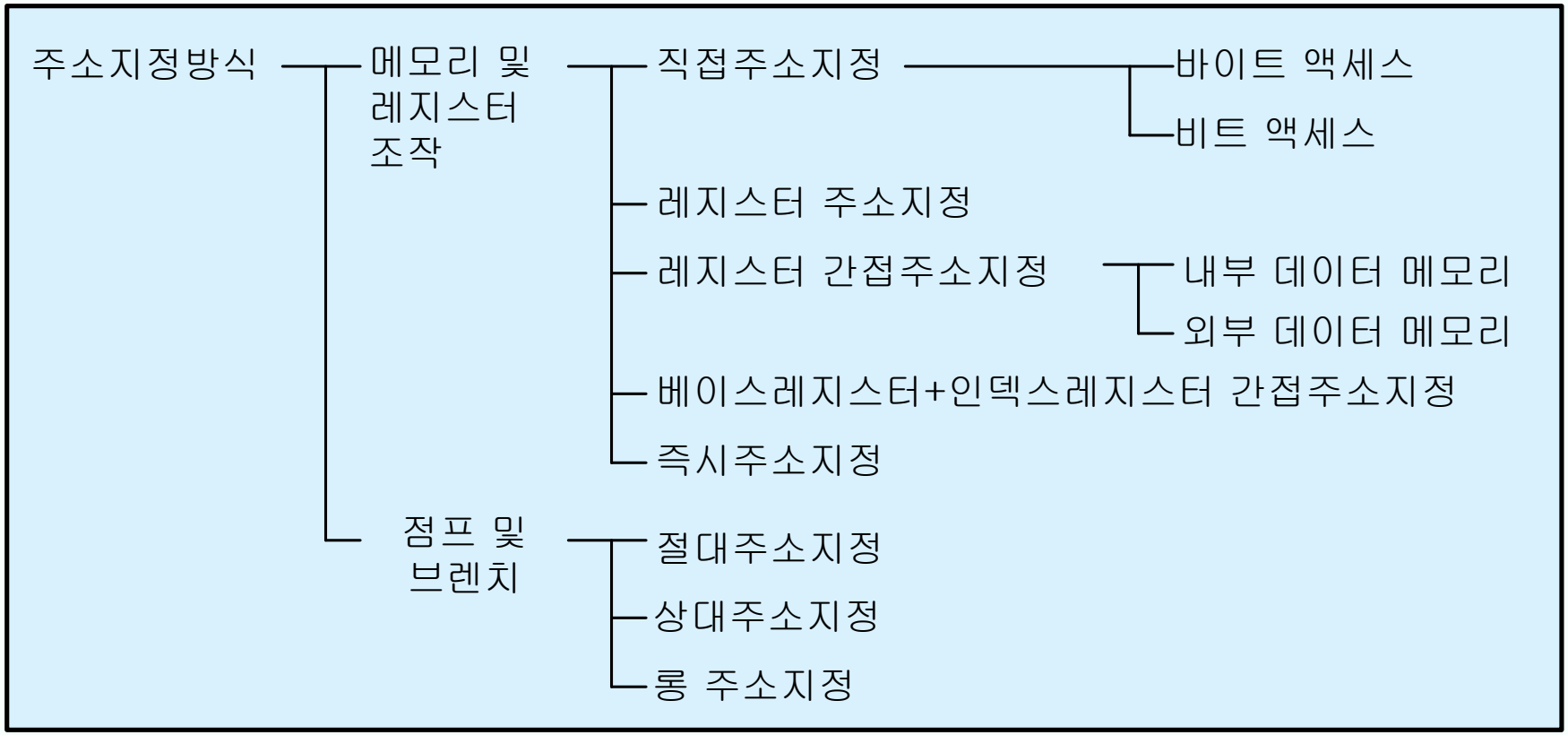
명령어	명령코드	오퍼랜드	동작
MOV A,#21H	74H	21H	(A) ← 21H
XCH A,3AH	C5H	3AH	(A) ↔ (3AH)

3바이트 명령어

명령어	명령코드	오퍼랜드1	오퍼랜드2	동작
MOV 40H,#50H	85H	40H	50H	(40H) ← 50H
JMP 2000H	02H	20H	00H	(PC) ← 2000H

3. 주소지정방식

- 명령어는 **Addressing Mode**로 구분하며, 이것은 **Operand**를 선택하는 방법을 의미한다.

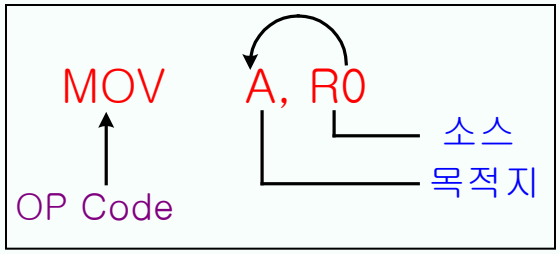


8051의 주소 지정 방식의 분류

4. 명령어 세트

- **데이터 전송 명령 (Data Transfer Instructions)**
- 산술연산 명령 (Arithmetic Operation Instructions)
- 논리연산 명령 (Logic Operation Instructions)
- 비트 조작 명령 (Bit Manipulation Instructions)
- 브랜치(프로그램 분기) 명령 (Branch Instructions)
- 부 프로그램 명령 (Subroutine Call Instructions)

4.1 데이터 전송 명령



❑데이터 전송 명령어 분류

① 내부 데이터 메모리와의 데이터 전송

② 외부 데이터 메모리와의 데이터 전송

③ 외부 프로그램 메모리와의 데이터 전송

❑ 내부 데이터 메모리와의 데이터 전송

❖ 즉치 주소 지정 (immediate addressing mode)

- 프로그램을 만들 경우 프로그램에서 사용하는 변수가 아닌 정수 데이터를 소스 오퍼랜드로 사용하는 경우에 정수는 명령어의 일부분
- 레지스터 혹은 내부 메모리에 저장될 데이터가 명령어에 직접 포함되는 형태

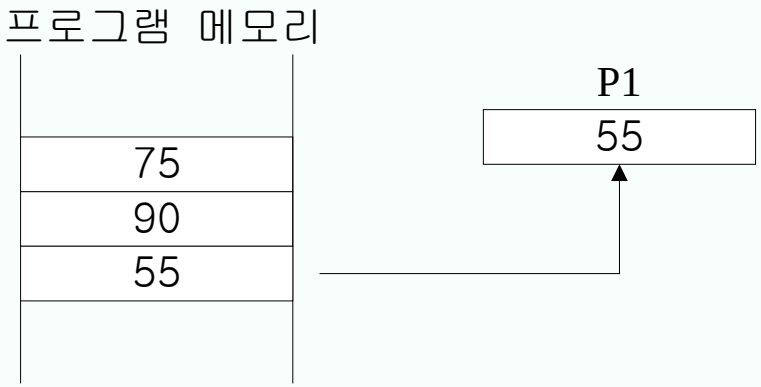
```
MOV     A,#data                 ; A ← 8비트 데이터
```

[Example 1] MOV A, #33H ; H는 16진수를 의미

※ #는 데이터를 표시,

※ MOV A, 33H ; 33H 번지의 내용을 어큐뮬레이터로 전송

[Example 2] MOV P1, #55H ; P1의 주소는 90H



❖ 레지스터 주소 지정 (register addressing mode)

- 8051에서는 각 뱅크에 있는 8개의 레지스터 R0~R7을 직접 액세스 가능
- 레지스터 번지 지정은 R0~R7 혹은 A 레지스터와 데이터 전송을 하는 것
- 8051에는 4개의 뱅크에 각각 R0~R7 이 있어서, 총 32 개의 레지스터가 있지만, 프로그램 실행 시점에서 보면 한번에 한 뱅크만 사용한다.

```
MOV     Rr,A            ; Rr ← A
MOV     A,Rr            ; A ← Rr
```

※ Rr = R0~R7

```
[Example 1]   MOV     PSW, #00010000B ; B는 2진수를 의미
              MOV     A, #30H         ; Acc에 30H저장
              MOV     R1, A           ; R1에 Acc의 내용을 저장
```

※ 8051은 4개의 뱅크가 있으며, 뱅크의 선택은 PSW레지스터를 사용하여 지정

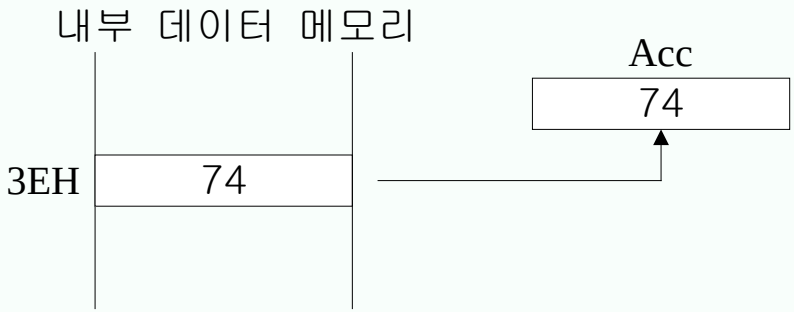
❖ 직접주소 지정 (direct addressing mode)

- 오퍼랜드로 사용되는 내부 데이터 메모리의 어드레스를 명령어에 직접 사용
- 내부 데이터메모리 **128** 바이트와 특수기능 레지스터 전체를 직접주소지정을 사용하여 액세스할 수 있다.

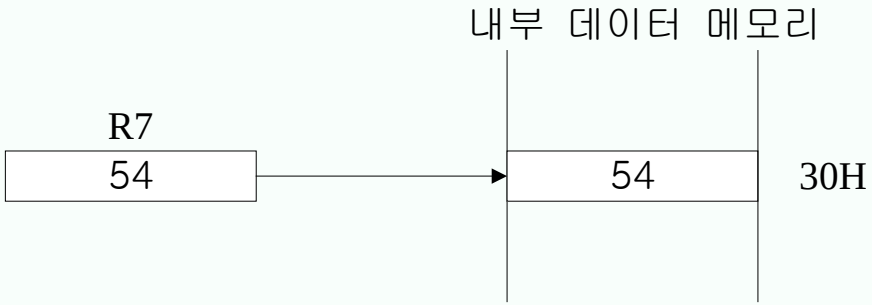
```
MOV       A,addr                ; A ← (addr)
```

※ addr=직접 번지 지정(00H～0FFH)

[Example 1] MOV A, 3EH ; 3EH번지의 내용을 Acc에 저장



[Example 2] MOV 30H, R7 ; R7의 내용을 30H번지에 저장



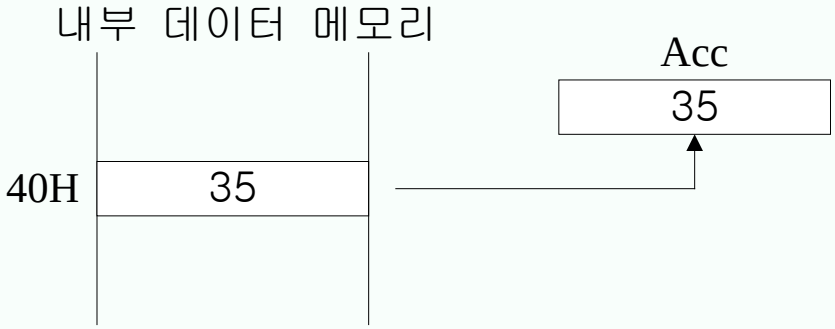
❖ 내부 데이터 메모리에 대한 레지스터 간접주소 지정 (internal data memory register indirect addressing mode)

- 내부 데이터 RAM에 대한 간접 번지 지정에서는 각 뱅크에 있는 R0, R1을 이용하여 내부 데이터 RAM의 번지를 가리키는 포인터로 동작시킬 수 있다.
- 어셈블리어에서는 간접 번지 지정을 의미하기 위해서 R0, R1앞에 @를 붙인다. (예: @R0, @R1)

```
MOV     @Ri,A          ; @Ri ← A
MOV     A,@Ri          ; A ← @Ri
```

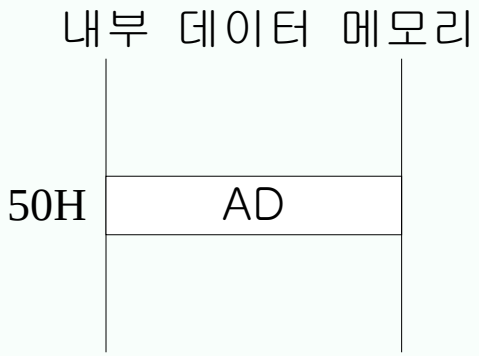
※ Ri = R0 혹은 R1, addr = 직접 번지 지정(00H~0FFH)

[Example 1] MOV R0, #40H ; R0에 #40H 저장
MOV A,@R0 ; R0가 가리키는 곳(40H)의 내용을 Acc에 저장



※ "@"가 간접주소 지정임을 나타낸다.

[Example 2] MOV R1, #50H ; R1에 데이터 50H를 저장
MOV @R1, #0ADH ; R1이 가리키는 곳(50H)에 #0ADH 저장



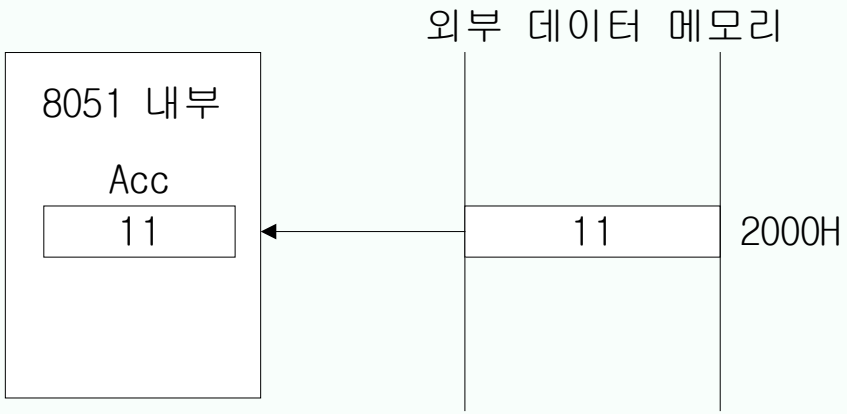
❖ 외부 데이터 메모리에 대한 레지스터 간접주소 지정 (external memory register indirect addressing mode)

- 8051에서는 외부 데이터 메모리를 64K(0000H~FFFFH) 바이트까지 확장 가능하며, 외부 데이터 메모리와 데이터를 주고 받으려면 어드레스 포인터 필요.
- 각 뱅크에 있는 R0, R1과 16 비트 레지스터 DPTR을 이용하여 외부 데이터 RAM의 번지를 가리키는 포인터 레지스터로 사용
- 이때 사용하는 명령은 앞에서 설명한 명령과는 달리 MOVX(X ; eXternal)를 사용하며, 소스와 목적지로는 CPU 내부의 A(어큐뮬레이터) 만 사용

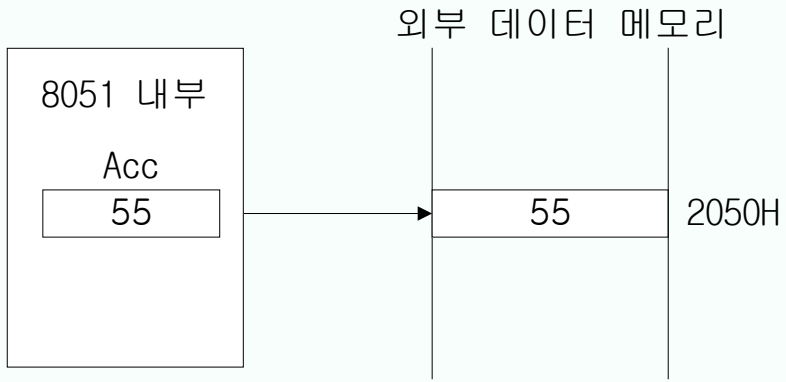
```
MOVX    @Ri,A          ; @Ri ← A
MOVX    A,@Ri          ; A ← @Ri
```

※ Ri = R0 혹은 R1

[Example 1] MOV DTPR, #2000H ; DPTR이 가리키는 곳(2000H 번지)의
MOVX A, @DPTR ; 내용을 Acc에 저장



[Example 2] MOV DPTR, #2050H ; Acc의 내용을 DPTR이 가리키는 곳
MOVX @DPTR, A ; (2050H 번지)에 저장

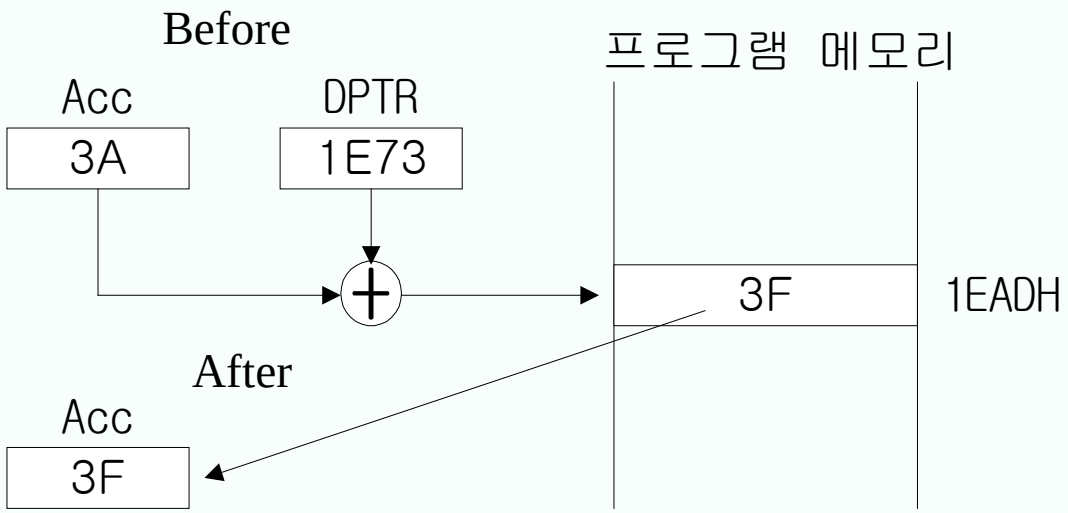


외부 프로그램 메모리와 데이터 전송(Indexed addressing mode)

- **8051**에서는 프로그램 메모리를 최대 **64K** 바이트까지 확장
- 보통 프로그램 메모리에는 프로그램만 저장되어 있는 것이 아니라 프로그램의 실행에 필요한 데이터도 포함
- 이 데이터를 **Read**할 때, 포인터로서 **16**비트 레지스터 **DPTR**과 **PC(Program Counter)**를 사용
- **DPTR**과 **PC**를 베이스 레지스터라 부르고, 인덱스 레지스터로는 **A**(어큐뮬레이터) 레지스터가 사용
- **A** 레지스터는 인덱스 레지스터로 사용해서 **DPTR, PC**가 가리키는 곳부터 오프셋**(offset)**을 나타내는데 사용하며 명령의 실행 결과가 저장

```
MOVC    A, @A+DPTR          ; A ← @A+DPTR
MOVC    A, @A+PC            ; A ← @A+PC
```

[Example 1] MOVC A, @A+DPTR ; DPTR+A가 가리키는 곳의 내용이 A로 저장



❖ 절대 점프 (absolute addressing mode)

AJMP	<11 bit address>

[Example 1] AJMP 01D2H

❖롱 점프(long addressing mode)

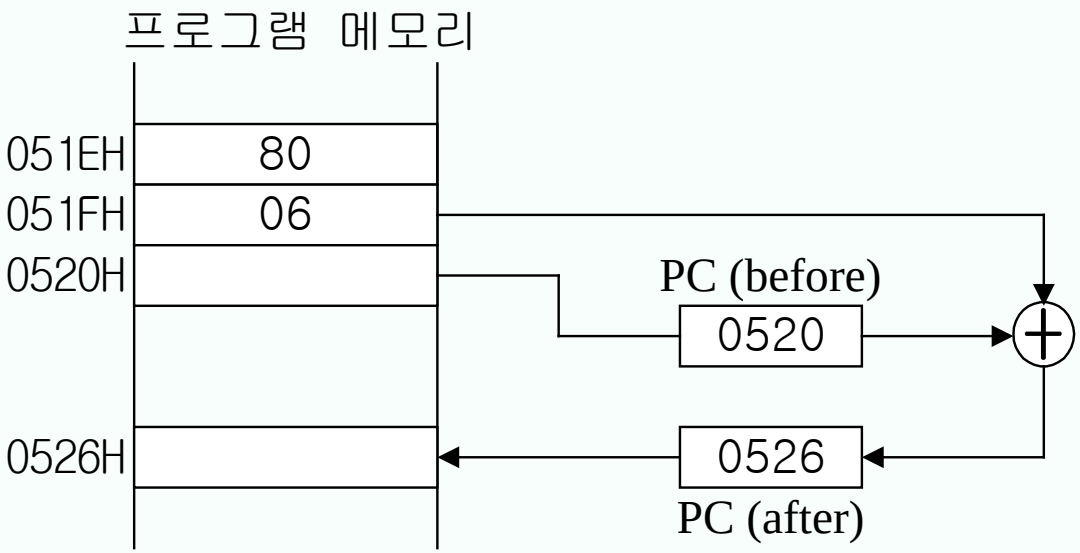
LJMP	<16 bit address>

[Example 1] LJMP 0A3EH

상대 점프(relative addressing mode)

SJMP	<Offset >

[Example 1] SJMP 06H



직접주소지정방식(direct addressing mode)

- 내부 데이터 메모리와 어큐뮬레이터(Acc) 사이
- 내부 데이터 메모리와 레지스터 사이
- 내부 데이터 메모리와 내부 데이터 메모리 사이

```
MOV    A, 50H
```

레지스터 주소 지정방식(register addressing mode)

- 명령코드가 필요로 하는 데이터가 레지스터 내에 저장되어 있는 경우

```
MOV    A, R1
```

레지스터 간접주소 지정방식(register indirect addressing mode)

- 레지스터 내에 저장된 값이 명령코드가 필요로 하는 데이터가 저장된 장소의 주소를 가지고 있는 경우
  - ① 내부 데이터 메모리에서의 데이터 전송(예, MOV A, @R1)
  - ② 외부 메모리와의 데이터 전송(예, MOVX A, @DPTR)

인덱스 레지스터 간접주소 지정방식 (Indexed addressing)

- 명령코드가 필요로 하는 데이터가 외부의 프로그램 메모리에 있는 경우

MOVC A, @A+DPTR

즉치 주소 지정방식(immediate addressing mode)

- 명령코드가 필요로 하는 데이터가 상수값으로 직접 주어지는 경우

MOV A, #1FH

상대 주소 지정방식(relative addressing mode)

- 분기(branch)에 관련된 주소지정방식이며, 현재위치로부터 (오프셋+2)만큼 떨어진 곳으로 점프하는 주소지정방식이며 프로그램 카운터(PC)의 값이 변한다.

SJMP 05H

절대주소 지정방식(absolute addressing mode)

- 절대주소(absolute address)로 점프하는 방식으로서 프로그램 카운터의 하위 11비트 값을 니모닉 다음에 위치하는 오퍼랜드로 대치시킨다

AJMP 01A5H

❑ 롱주소 지정방식(long addressing mode)

- 절대주소 지정방식의 일종으로서 64K 바이트의 전 메모리 영역으로 점프를 가능하게 하는 명령어이며, 점프할 주소는 16비트로 표현된다.

LJMP 8000H

참고자료

- 어셈블러로 배우는 마이크로프로세서 8051기반의 이론, 실습, 구현
  - 한빛미디어
- 마이크로프로세서
  - 순천향대학교 정보기술 공학부 이상정 교수님
  - http://sjlee.sch.ac.kr/lecture/mp/06-1-mp.htm