제 5장 사칙연산 명령

가감승제의 사칙연산을 할수가 있읍니다.
대부분의 8비트CPU 에 서는 곱셈 나누셈을 할수가없었는데 사칙연산을 할수가 있게 된 것은 16 비트 CPU 의 큰 특징입니다.

ADD 명령의 사용법:
```
ADD     AX,1234H
```
현재의 AX의 내용에 숫자 1234H 가 더하여서 그 결과를 AX 레지스터에 다시 저장
```
        범용레지스터(8/16비트)     범용 레지스터(8/16비트)
ADD     메모리      (        )      메모리      (        )
                                   숫자        (        )
```
양쪽 다 메로리로 조합되는것은 허용되지않음니다.
*.주의 : 메모리에 수치를 더할때에는 BYTE 혹은 WORD 지정이 필요
```
 ADD    [BX],12H          > ADD   BYTE PTR [BX],12H
```
이렇게 하여 바이트 혹은 워드를 지정하지 않으면 안된다.
*.주의 :
```
AAA     EQU     32H   는 AAA 가 상수이므로   숫자   이다.
BBB     DW    5678H 는 BBB가 변수이름 이므로 메모리를 나타내는 간접방식의 일종
```

4040H+0102H 를 더하고 그결과 4142H 의 41H ,42H에 해당하는 문자를 출력한다.
```
MAIN    SEGMENT
        ASSUME  CS:MAIN
;
        MOV     BX,4040H
        ADD     BX,0102H
        MOV     DL,BH           상위 바이트
        MOV     AH,2
        INT     21H
        MOV     DL,BL           하위 바이트
        MOV     AH,2
        INT     21H
;
        MOV     AH,4CH          종류
        INT     21H
;
MAIN    ENDS
        END
```

결과 :
```
A>ADD1
        AB
      A>
```
아스키 코드 41H 에 대응하는 문자 A" 와 42H 에 대응하는 문자 B"가 표시
직접 숫자를 숫자로 출력하는 방법은 없는것인가 ? ---
 MS-DOS시스템호출에 숫자를 직접 출력하는 방법이 준비되어 있지않읍니다.

키입력 방법

키보드로 부터 한문자를 입력할려면 ,MS-DOS 의 펑션호출(function)의 1 번을 사용합니다. AH 레지스터에 1 을 설정하고 펑션 호출을 수행하면, 키보드로부터 입력이 있을때 까지 기다리고 있다가 ,입력된 문자의 아스키코드를 AL register 로 돌려 보내줍니다.

```
   MOV     AH,1
   INT     21H
   AL          > 입력된 문자의 아스키코드


MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
ONE     EQU     1
TOLOWER EQU     'a'-'A' ;상수나 변수의 정의 속에서 덧셈 뺄셈을 사용할수있다
;
        MOV     AX,MAIN     DS 설정
        MOV     DS,AX
;
        MOV     AH,1        펑션호출로 키입력된 값이 AL에 저장된다.
        INT     21H
        MOV     KEEP,AL     1문자입력하여 결과를 KEEP라는 변수에 저장
        ADD     AL,ONE     ; 입력된 데이터에 1을 더하고
        MOV     DL,AL       아스키 코드의 문자를 출력
        MOV     AH,2
        INT     21H
;
        MOV     DL,KEEP     2를 더한 아스키 코드의 문자를 출력
        ADD     DL,TWO
        MOV     AH,2
        INT     21H
;
        MOV     DL,KEEP     20H를 더하여 소문자로 변환하여 출력
        ADD     DL,TOLOWER
        MOV     AH,2
        INT     21H
;
        MOV     AH,4CH
        INT     21H          종료하고 OS로 돌아간다
;
TWO     DB      2
KEEP    DB      ?       ;단순히 데이터를 저장하기위하여
MAIN    ENDS
        END
```

a'-'A'는 a 와 A 의 아스키 코드의 차이를 나타낸다.
'a' 에 대응 하는 아스키 코드 61H 로 부터 A' 에 대응하는 41H 를 뺀
61H - 41H = 20H 를 의미 합니다. 따라서 TOLOWER 에는 숫자 20H 가 저장

디버거의 사용 예와 역워드 형식

예제
1000H + 1234H를 계산하여 ANS1에 저장한다.

```
AMIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
        MOV     AX,CS
        MOV     DS,AX
;
        MOV     AX,1000H
        MOV     BX,1234H
        ADD     AX,BX
        MOV     ANS1,AX
;
        MOV     AH,4CH
        INT     21H
;
ANS1    DW      0000            > MOV   [0013],AX 로 역어셈블된다,즉 변수ANS1은
;                                 0013H 번지 할당되어 있기 때문에
MAIN    ENDS
        END
```

DEBUG 는 EXE 및 COM 형 실행 화일 을 로드 하였을 때에는 실행할수있는 상태로 레지스터 종류를 세트하여 대기하고 있읍니다
그외의 화일을 로드 하였을 때는 단순히 로드하였을때의 화일의 선두 번지를 CS:IP 에 넣고 대기 하고 있읍니다.

역어셈블된 ANS1 변수의 번지 0013 을 보면 3422 가 들어 있다
역어셈블 결과의 XOR AL,22 라는 것은 단순히 이 데이터를 명령으로보고 역어셈블해 버린것이므로 아무의미가 없다
원래의 연산결과는 1000H + 1234H = 2234H 라고 될터인데
0013H 번지와 0014H 번지에는 3422 로서 저장되어 있다는 것입니다.
즉,상위와 하위가 서로 바뀐 형태로 결과가 저장되어 있음니다.
80 계열CPU의 특징적인 현상으 로 ,16비트 레지스터 상의 데이터(워드데이터)를 메모리에 전송할때에는 상위 하위 바이트가 바뀐다라는 규칙이 있기 때문입니다.
이와 같은 전송방식을 역워드 형식 이라고 합니다.
메모리 상에 데이터를 저장할 때에는 상하 바이트가 바뀌므로 다시 이것을 레지스터 상 에 전송할때에는 ,또한 상하 바이트가 바뀌어서 전송되기 때문에 바른 형태로 데이터를 전송할수가 있읍니다.
따라서 어셈블러를 사용하여 프로그램을 작성할때에는 역워드로 되는 것은 별로 의식할필요가 없읍니다.

자리올림(carry)과 ADC 명령

16비트 레지스터에 저장되지 않는 것을 미리 알고 있는 경우에는
2개의 16비트 레지스터를 사용하여 32비트 숫자를 표시 ,하위 16비트 를 덧셈할때에
발생한 자리올림(carry) 을 어딘가에 기억해 놓고 ,상위 16비트를 덧셈할때 더해주
는 방법을 취합니다
자리올림을 기억하는 플래그 레지스터 중의 캐리 플래그(carry flag)비트
덧셈을 할때에 캐리 플레그를 동시에 더하는 ADC 명령(ADdition with Carry)

```
MAIN    SEGMENT
        ASSDUME CS:MAIN,DS:MAIN
;
        MOV     AX,MAIN
        MOV     DS,AX
;
        MOV     AX,1000H
        MOV     BX,8000H
        MOV     CX,2000H
        MOV     CX,8123H
        ADD     BX,DX       ;두개의 16비트의 합을 구한다
        ADC     AX,CX       ;위에서 생긴 캐리까지 고려하여 상위16비트합을구한다.
        MOV     ANS1,AX
        MOV     ANS2,BX
;
        MOV     AH,4CH
        INT     21H
;
ANS1    DW      0
ANS2    DW      0
;
MAIN    ENDS
        END
```

데이터를 교환하는 XCHG 명령

XCHG (exchange) 명령은 2 개의 데이터를 교환하는 명령
직접숫자를 교환할수는 없읍니다.
메모리끼리의 교환도 불가능합니다.
첫번째 두번째 오퍼랜드는 일치되어야 합니다.

```
XCHG    범용 레지스터 (8/16비트)        범용 레지스터 (8/16비트)
        메모리        (8/16비트)   ,    메모리        (8/16비트)

MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:DATA
;
        MOV     AX,DATA
        MOV     CS,AX
;
        MOV     AX,1000H
        MOV     BX,8000H
        MOV     CX,2000H
        MOV     CX,8123H
```

```
        ADD     BX,DX           ;두개의 16비트의 합을 구한다
        ADC     AX,CX           ;위에서 생긴 캐리까지 고려하여 상위16비트합을구한다.
        XCHG    AL,AH
        XCHG    BL,BH
        MOV     ANS1,AX
        MOV     ANS2,BX
;
        MOV     AH,4CH
        INT     21H
;
MAIN    ENDS
;
DATA    SEGMENT
ANS1    DW      0
ANS2    DW      0
DATA    ENDS
        END
```

이번에 주의 하지 않으면 안되는 것은 ,지금까지와 달라 서 데이터 세그먼트가 별도로 설정되어 있다는 사실입니다.
이 경우에는 어셈블할 때에는 데이터 세그먼트의 위치는 결정되지 않고 ,링커에 의해 상대적인 배치가 정해진 다음 실행할때에 절대 번지가 결정됩니다.
따라서 어셈블할 때에 MOV AX,DATA 에 대응 하는 부분은 B8 --- R 로되어 있으므로 메모리 사에 로드 하였을 때에는 MOV AX,2245 와 같이 값이 결정되어 있읍니다.
자, 여기에서 데이터를 볼려면 세그먼트 베이스와 오프셋 번지를 같이 지정하여 내용을 표시 하지 않으면 안됩니다.
역어셈블한 결과 데이터 세그먼트에 세트되는 값은 ,2245H 라는 것을 알수있읍니다.

```
_D2245:0,F
2245:0000  00 00 00 00 F4 75 10 8B - 7E EE 39 3E 42 1F 76 07
.....tu..~n9>B.V.
```

여기서 D 는 데이터를 보이라는 명령 (DUMP)
0,F 오프셋 번지 지정
2245 는 데이터 세그먼트의 베이스 번지가 2245 이다.

SUB 명령의 사용법(뺄셈 명령)

SUB 범용레지스터(8/16비트) 범용레지스터 (8/16비트)
메모리 (8/16비트) 메모리 (8/16비트)
, 숫자 (8/16비트)

메모리 자체끼리는 뺄셈을 할수없다:
SUB (subtract) 명령은 첫번째 오퍼랜드로 부터 2번째 오퍼랜드 의 내용을 뺀다음 결과를 첫번째 오퍼랜드에 저장합니다.

예제) SUB1.ASM
변수 AAA 의 200 에서 100 을 빼는 프로그램

```
MAIN    SEGMENT
        ASSUNE  CS:MAIN,DS:MAIN
;
        MOV     AX,MAIN
        MOV     DS,AX
        MOV     AL,100   ; 16진수로  64H
        SUB     AAA,AL
;
        MOV     AH,4CH
        INT     21H
;
AAA     DB      200  ; 16진수로  C8H
;
MAIN    ENDS
        END
```

어셈블러에서는 끝에 숫자의 표기법을 나타내는 (H) 나 (B) 가 붙어 있지않으면
내정적으로 10진수로써 취급합니다.

SBB 명령(subtract with borrow)

캐리와 함께 뺄셈을 하는 SBB 명령:
자리 빌림이 발생하면 캐리 플레그가 세트된다.
사용법:
1) SUB명령에 의해 하위 16비트(혹은 8비트)를 뺄셈
2) SBB명령에 의해 상위 16비트(혹은 8비트)를 뺄셈
만일 처음부터 16비트 연산에 SBB명령이나 ADC 명령을 사용하면
캐리 플래그가 이전부터 가지고 있었던 연산과 직접 관계없는 값까지 뺀다든지
더해버리는 경우가 있기 때문입니다.

예제) SUB2.ASM
12340000H - 1000H 를 계산하는 프로그램

```
MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
        MOV     AX,MAIN
        MOV     DS,AX
;
        MOV     WORK1,1234H
        MOV     WORK2,0000H
        MOV     WORK2,1000H
        SBB     WORK1,0000H
;
        MOV     AH,4CH
        INT     21H
;
WORK1   DW      0
WORK2   DW      0
;
MAIN    ENDS
        END
```

여기 프로그램에서 하려고 하는 계산을 식으로 나타내면

```
    12340000H
 -      1000H
-------------
    1233F000H
```

라는 것이 됩니다.
WORK1 이 상위 16비트 ,WORK2 가 하위 16비트로 메모리에 직접 숫자를 대입하고 ,뺄셈을 하고 있읍니다. 여기에서 연산하기 전에 WORD PTR 이 붙어 있지 않았는데 ,이것은 변수이름이 DW 에 의해 명확하게 WORD 형의 속성을 갖고 있기 때문입니다.
이와같이 변수에 숫자를 대입할 경우에는 형(type) 지정이 필요없읍니다.
다만 디스어셈블할 때에는 WORD PTR 이 붙습니다.

예) SUB3.ASM
변수를 경제적으로 PTR 지정에 의해 지정된 형(type)으로 사용한 예

```
MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
        MOV     AX,CS
        MOV     DS,AX
;
        MOV     AH,11H
        MOV     BH,22H
        MOV     CL,33H
        MOV     SI,4444H
        MOV     BP,5555H
        ADD     AH,66H
        MOV     BYTE PTR ANS1,AH
        ADD     CL,0FFH
        ADC     BH,0H
        MOV     BYTE PTR ANS2,BH
        MOV     BYTE PTR ANS2+1,CL
        SUB     BP,SI
        MOV     WORD PTR ANS3,BP
;
        MOV     AH,4CH
        INT     21H
;
ANS1    DW      0
ANS2    DW      0
ANS3    DB      0
ANS4    DW      9999H
;
MAIN    ENDS
        END
```

선언된 형 이외에서의 변수의 사용법

앞에서 ANS1,ANS2 등과 같은 것은 번지를 나타내는 이름
이들 변수이름은 선언된 형(바이트 인지 워드 인지)을 속성으로 서 지니고 있으므로 그대로 사용하였을 때에는 그 선언된 형 이외의 형태로 사용하고 싶을때에는 형 + PTR 을 붙임으로써 강제적으로 다른 형으로 사용할수가 있읍니다.

```
MOV     BYTE PTR ANS2+1,CL
```

라는 부분은 :

변수이름에 대한 덧셈 뺄셈은 ,번지의 값을 덧셈,뺄셈한다.
(ANS2 가 나타내는 번지 +1)
그러나 실제로느 이와 같은 사용법은 별로 추천하지는 않습니다.특수한경우를 제외하고는 변수는 선언된 형으로 취급하도록 하는 것이 현명합니다.

음수와 보수 표현

보수(complement)의 개념:
10진수로 100을 기준으로 한경우의 5의 보수는 100-5=95,따라서 5의 보수는 95이다.
16진수의 1바이트 = 8비트숫자로 0 ~ 255 (= 2^8 - 1 = 0FFH)까지의 숫자를 취합니다.
2^8(=100H) 를 기준으로 한 보수를 생각 합니다.
2의 보수는 100H - 2H = 0FFH
2^8을 기준으로 한 3의 보수 100H - 3H = 0FDH
여기에서 -1을 나타내는 데에 1의 보스를 사용하여 0FFH
          -3을 나타내는 데에 3의 보스를 사용하여 0FDH 등으로 표현
예)
1+(-1)=01H+0FFH=100H > 0
1+(-3)=01H+0FDH=0FEH > -2
여기에서 주의사항 : 보수 표현 을 사용한 경웅에 어느수가 음수를 나타낸것인가를
                    결정하지 않으면 안된다는 것입니다.
아무표시도 안하면 0FDH 가 정수의 253 인지,음수의 -3인지 알수가 없다.
그래서 보수표현을 사용한 경우의 약속으로서 그숫자를 2진수로 나타냈을때,
최상위 비트(MSB)가 0 인 것을 양수,최상위 비트가 1 인것을 음수로 간주하도록 정하고 있읍니다.
이렇게 하면, 8비트 숫자에서는 0 ~ 7FH 까지가 양수,
                          80H~ 0FFH 까지가 음수(-80H ~-1H 에 대응)
                이것에 의해 -80H~ +7FH 까지의 숫자를 나타낼수가 있읍니다.
16비트 숫자를 취급하는 경우에도 마찬가지로 2^{16}=10000H를 기준으로
                          0~7FFFH 까지가 양수 8000H~ 0FFFFH까지가 음수
MSB : most significant bit 의 약자이고 최상위 비트를 표시함
LSB : least significant bit 의 약자이며 최하위 비트를 말한다.

                MSB   LSB
8비트의 경우
                >
                   7 6 5 4 3 2 1 0
               MSB   LSB
16비트의경우
                >
                   1514131211109 8 7 6 5 4 3 2 1
*.주의)
보수표현을 사용하여 -80H ~7FH 까지의 수를 취급할 것인가,보수표현을 사용하지않고 0~0FFH까지의 숫자를 취급할것인가는 프로그래밍을 하는 사람이 정하는 것으로서 같은 숫자가 양수르 의미하는 것인가 ,음수를 의미하는 것인가는 경우에 따라 다르다 라는 것입니다.
취급하는 숫자가 양수뿐이고 뺄셈을 하지 않는 경우에는 보수표현을 사용할필요가 있읍니다
어느것을 사용할것인가는 프로그램의 목적에 따라 결정하지 않으면 안됩니다.
어셈블러에서는 양수 음수가 어떤식으로 해석되는가를 봅시다.
원래 숫자가 1H 및 0FFH인 경우 그숫자가 그대로 레지스터로 전송되는 숫자가 됩니다.
원래의 숫자가 -1H 및 -0FFH 는 , 단순히 보수로 변환되어서 전송하는 숫자로 됨.
8비트 레지스터로 -0FFH를 전송하는 경우: 1H 로바른 변환이 아니다

어셈블러에서는 보수표현이 가능한 상하의 범위의 검사는 하지않으므로 주의가 필요함
-0FFH를 바르게 취급하려고 한다면 8비트 레지스터로는 불가능하고 ,16비트 레지스터를 사용할필요가 있읍니다 .
16비트 숫자는 메모리상으로는 역워드 형식으로 기억되어 있으므로
MOV AX,1234H
B8 1234 가 아니라 B8 34 12 로 나열되어 있음니다.

부호 확장 명령

40H와 50H 를 더하면 결과는 90H 가 되는데 ,이것은 보수 표현에서는
-70H 라는 음수에 해당 되므로 바른 결과 가 아닙니다.
(10010000 의 2의 보수는 우선 1의 보수로서는 01101111 이되고
01101111+1=01110000=70H 가 된다.최상위 비트가 1이므로 음수 -70H 가 된다.)

ADD DL,AL >가능
ADD DX,AL >레지스터 불일치 따라서 불일치
레지스터의 전송이나 덧셈 뺄셈에서는 2개의 ,레지스터의 크기가 같지 않으면 안됨

양수의 덧셈:
모든 수를 양의 정수로서 취급하는 경우 - 8비트 레짓스터의 값을 16비트로 확장한 다음 더하면 됩니다.
MOV AH,1 한 문자 입력 루틴 결과는 AL 레지스터에 놓여진다.
INT 21H
MOV AH,0 > AL 레지스터의 상위 8비트를 클리어 한다.
ADD DX,AX
양수만을 취급할때에는 상위 바이트에 00을 대입한다.
AL =41H

AX 한문자 입력후의 각레지스터의 상태
(A ″=41H 를 입력한 경우)

AH	AL
미정	41

===== AH 레지스터에 0 을 대입 8비트 데이터를 16비트 데이터로 확장 =====
AX=0041H

AX 한문자 입력후의 각레지스터의 상태
(A ″=41H 를 입력한 경우)

AH	AL
00	41

예제) SGN2.ASM
4개의 숫자 50H,60H,80H,F0H를 합하여 ANS에 저장한다.

```
MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
        MOV     AX,CS
        MOV     DS,AX
;
        MOV     AX,0
        MOV     DX,0
        MOV     DL,VAR1
        ADD     AX,DX
        MOV     DL,VAR2
        ADD     AX,DX
        MOV     DL,VAR3
        ADD     AX,DX
        MOV     DL,VAR4
        ADD     AX,DX
        MOV     ANS,AX
;
        MOV     AH,4CH
        INT     21H
;
        VAR1    DB      50H
        VAR2    DB      60H
        VAR3    DB      80H
        VAR4    DB      F0H
        ANS     DW      ?
;
MAIN    ENDS
        END
```

디버거 상에서 실험하여 확인해 볼것(HWP에서 F3지울영역설정,ctrl-E 지우기 실행,
예제를 아스키로 세이브 MASM 실행)
이방법은 256이란 수치까지는 안심하고 덧셈을 할수가 있읍니다.

보수표현에 의해 많은 데이터를 더하는 법

음수는 상위 바이트에 00을 대입하기만 하는 것으로 는 안됨,
(음)95H　　>　0095H(양)
(음)F7H　　>　00F7H(양)
따라서 음수를　8비트에서 16비트로 변환할때에는
(음)95H　　>　FF95H(음)
(음)F7H　　>　FFF7H(음)
(음)80H　　>　FF80H(음)
양수이면 상위 바이트에 00을 대입 ,음수이면 상위 바이트에 FF를 대입한다.

이와 같은 작업을 일일이 프로그램을 작성하여 하려고 한다면 대단히 곤란할것 입니다.그래서 위에서 기술한 규칙에 따라 8비트 숫자를 16비트로 변환해주는 명령으로서 **CBW** (convert byte to word) 명령을 만들어 놓았읍니다.
이것을 **부호확장명령** 이라고 합니다.

보수표현을 사용하여 양수 음수를 사용하는 경우에는
 최상의 비트가 0 인 것을 양수, 1 인것을 음수라고 한다.
8비트 수치를 16비트 수치로 확장할때에:
 최상의 비트가 0 이면 상위 8비트 모두에 0을 대입,
 최상의 비트가 1 이면 상위 8비트 모두에 1을 대입

```
          양수의 확장            음수의 확장
8비트           0XXXXXXXX              1XXXXXXX

        V       V              V       V
16비트  000000000XXXXXXX       111111111XXXXXXX
```

CBW ....AL레지스터의숫자를 부호확장하여 AX 레지스터에 저장한다.
CWD ....AX레지스터의 숫자를 부호확장하여 ,상위16비트를 DX레지스터에 ,
        하위 16비트를 AX 레지스터에 저장한다.
CWD (convert word to double word) 32비트 레지스터로 간주하는 명령

부호 확장명령의 사용법:
예제) SGN3.ASM
부호가 붙은 4개의 숫자를 합하여 ANS 에 저장하는 프로그램

```
MAIN    SEGMENT
        ASSUME  CS:MAIN,DS:MAIN
;
        MOV     AX,CS       DS 초기설정
        MOV     DS,AX
;
        MOV     AX,0        AX,DX 를 0으로 클리어
        MOV     DX,AX
        MOV     BX,OFFSET VAR ;변수이름 VAR이 가리키는 오프셋 번지 를 BX에 저장
        MOV     AL,[BX]       ;BX에 들어있는 번지값의 내용을 AL 에 넣는다
        CBW                   ;부호 확장
        ADD     DX,AX                                                  ^
        MOV     AL,[BX]+1     번지의 표기법에는 여러종류가있다.
        CBW
        ADD     DX,AX
        MOV     AL,[BX+2]               C 언어의 포인트 개념과 같다.
        CBW
        ADD     DX,AX
        MOV     AL,3[BX]
        CBW
        ADD     DX,AX
        MOV     ANS,DX
;
        MOV     AH,4CH
        INT     21H
;
VAR     DB      50H,60H,80H,0F0H  ;데이터를 연속하여 할당시킨다.
ANS     DW      ?
;
MAIN    ENDS
        END
```

주의) CBW 명령은 AL 레지스터로부터 AX 레지스터로 밖에 변환할수없다.

VAR변수는 :(독립된 변수이름이 주어져 있지 않읍니다.)

```
VAR     DB      50H
        DB      60H
        DB      80H
        DB      0F0H
```

와 같이 지정하여도 똑같읍니다.
여기서 변수의 내용을 꺼내기 위해서 변수이름 VAR이 나타내는 번지로부터 상대의 위치를 지정할필요가 있읍니다.

```
MOV     BX,OFFSET VAR
```

(오프셋 의사명령에 의해 변수이름이 가지는 속성의 하나인 오프셋 번지의 값을 꺼낼수있다 .)

번지의 표기법중에서
[BX]+2
[BX+2]
2[BX] 는 완전히 같은 방법을 다른 방법으로 썼을 뿐이다.
디버거에서는 역어셈블할 때의 표기는 [BX+2]의 형식으로 되어있다.
단, 변수이름을 사용해도 마찬가지 입니다.
[BX]+1 은 VAR+1 등으로 치환할수가 있읍니다.
여기에서 VAR + 1 이라는 것은 변수의 내용에 1 을 더하는 것이 아니라
변수이름이 나타내는 번지에 1 을 더한다.
[BX+2] 는 VAR+2 로
3[BX] 는 3+VAR 로 라는 식으로 바꿀수가 있읍니다.

*.보수로 표시된 80H 가 왜 -80H 가 되는가.?
80H =1000 0000 이어서 MSB=1 이므로 음수이다. 우선 1 의 보수를 취하면 0111 1111.
여기에 +1 하면 1000 0000 즉 80H 가 되므로 -80H 값이다.

여기에서는 결과를 숫자로서 출력하는 루틴을 덧붙이지 않았다.
매번 사용하는 루틴은 한번 작성해두면 나중에 다른 프로그램을 만들때에 인용할수있도록 하기위해서 루틴을 다른 화일 로 작성해두고 INCLUDE 문으로 포함시키는 기능이나,분할 컴파일 하여 오브젝트 화일로 작성해두고 LINK 에 의해 링크하는 기능도 있읍니다.. 당분간은 결과를 디버거 를 사용해서 하도록합시다.