5.10 곱셈명령과 구조

곱셈명령 MUL(multiply) 에서는
8비트 * 8비트,혹은 16비트 * 16비트 를 계산할수가 있읍니다.
사용할때의 제한 :
우선 곱셈할 한쪽은 반드시 AL 레지스터(혹은 AX 레지스터)에 넣어두지 않으면 안됩니다. 결과는 AX 레지스터(혹은 DX 레지스터를 상위16비트,AX 레지스터를 하위 16비트로 간주한 32비트 레지스터)에 저장됩니다.
따라서 곱셈명령의 오퍼랜드는 하나 이다.

8비트 곱셈			곱셈결과 저장
AL	*	8비트의 레지스터 혹은 메모리	> AX

16비트 곱셈			곱셈결과 저장
AX	*	16비트의 레지스터 혹은 메모리	> DX AX

예) MUL BL
여기서 지정된 레지스터가 8비트인가 ,16비트인가에 따라서
자동적으로 AX 레지스터가 결정 됩니다.
부호없는 곱셈 MUL(multiply)
부호있는 곱셈 IMUL(integer multiply)

사용할때의 주의할점: 오퍼랜드가 메모리일때
예)

```
        MUL     DATA1
        ..............
DATA1   DB      20H
```

와같은 경우에는 DATA1의 형이 바이트 변수라고 알수있으므로 괜찮지만

```
        MOV     BX,OFFSET DATA2
        MUL     [BX]
        ..............
DATA2   DW      1234H
```

와 같은 경우,[BX]에 의해 표시된는 메모리가 바이트단위인지,워드단위인지가 불명확하므로 에러가 됩니다. 이와 같은 경우에는 [BX]에 의해 지정되는 메모리가 바이트 단위인지를 명시하기 위해서

```
MUL     WORD    PTR[BX]
```

와 같이 사용하지않으면 안됩니다.

또하나 주의 할점: 곱셈명령에서는 직접 숫자하고는 곱셈을 할수없다
예)

```
MUL     3
```

이와 같이 는 할수없으므로

```
MOV     BL,3
MUL     BL
```

와 같이 하도록 되어 있읍니다.
또한 상수이름은 수치로 변환되기 때문에 상수이름도 사용할 수 없습니다.

```
ABC     EQU     12H
        .........
        MUL     ABC
```

와 같이는 쓸수가 없읍니다.

예)MUL1.ASM
부호없는 두개의 메모리끼리,부호있는 두 메모리 끼리 곱셈 보기

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
        MOV     AX,CS  ;  CS와 DS를 같게 설정
        MOV     DS,AX
;
        MOV     AL,DATA_1A
        MUL     DATA_1B
        MOV     ANS1,AX
        MOV     AX,DATA_2A
        MUL     DATA_2B
        MOV     ANS2_A,DX ;결과의 상위,하위를 직접변수에 넣을 때에는 2개의 변수
        MOV     ANS2_B,AX ;로 나누어야만 한다.
;
        MOV     AL,DATA_3A
        MOV     BL,DATA_3B
        IMUL    BL
        MOV     ANS3,AX
        MOV     AX,DATA_4A
        MOV     BX,OFFSET DATA_4B
        IMUL    WORD PTR [BX]
        MOV     DI,OFFSET ANS4   이와 같이 지정하면 변수이름은 하나이면 된다.
        MOV     [DI],DX
        MOV     [DI+2],AX
;
        MOV     AH,4CH
        INT     21H
;
DATA_1A DB      0F0H
DATA_1B DB      11H
DATA_2A DW      1234H
DATA_2B DW      2001H
DATA_3A DB      -10H
DATA_3B EQU     11H    ;EQU 에 의한 정의는 어디에 있어도 상관없지만 주의가 필요
DATA_4A DW      -1000H
DATA_4B DW      1234H
;
ANS1    DW      ?
ANS2_A  DW      ?      ;16비트 숫자 곱셈은 DW로 정의 워드에 한번에 저장할수
ANS2_B  DW      ?      ;없으므로 ANS2-A 와 ANS2-B 로 나누어 저장
ANS3    DW      ?
ANS4    DD      ?       ;더블워드(4바이트)를 선언
;
CODE    ENDS
        END
```

DATA_1B 는 DB에서 정의된 **변수이름**이고 ,**바이트**형의 속성을 가지고 있으므로
BYTE PTR 의 지정은 생략할수있읍니다.(붙여도 상관없읍니다.)
DATA_2B 가 워드형의 속성을 갖고 있기 때문에 직접 곱셈을 할수가 있읍니다.
결과가 DX:AX 레지스터쌍에 의한 32바이트 레지스터에 저장되어 집니다.

이결과를 ANS2 로 전송하고 싶은데 한번에 할수없으므로 DX레지스터와 AX레지스터로 나누어서 전송합니다.
레지스터의 변수이름의 형이 일치하지 않으면 안되므로 ,ANS2 를 2개의 워드형의 변수 ANS2_A 와 ANS2_B 로 나누고 있읍니다.
변수이름은 속성으로서 번지 이외에도 바이트인지 워드인지를 구분하는 타입을 가지고 있는 것에 주의 해 주십시요
만일 ,

```
        MOV     ANS2,DX
        MVO     ANS2+2,AX
        ........
ANS2    DD      ?
```

ANS2는 더블 워드형(double word)의 속성을 갖고 있으므로 ,더블 워드형의 변수에 워드형의 레지스터값을 전송하려고 한것으로 되어서 ERROR 가 됩니다.
이와 같은 사용법을하고 싶다면 강제적으로 타임을 일치시키기 위해서

```
        MOV     WORD PTR ANS2,DX
        MOV     WORD PTR ANS2+2,AX

ANS     DD      ?
```

라고 하면 되는 것입니다.

전방 참조(forward reference)
매크로 어셈블러는 소스 프로그램을 기계어로 번역할 때 전체를 선두로부터 2번 반복해서 읽고 지나가면서 작업을 해 나갑니다.(그 때문에 2패스 어셈블러(two pass assembler) 라고 불리워진다.)
어셈블러가 첫번째 선두로 부터 읽어 내려 갈때에 ,DATA_3B는 프로그램 끝에 있기 때문에 아직 정의되어 있지 않으므로 ,이 시점에서는 아직 기계어를 결정할수가 없읍니다.그래서 어셈블러는 이부분의 기계어를 결정하지 않은 채로 생각할수 있는 최대 바이트인 4바이트를 확보해두고 다음으로 진행해 나갑니다.그리고 마지막쪽에서 DATA_3B가 상수 이고 그 값이 11H 라는 것을 알수있읍니다.그래서 2번째 번역 작업할때에 미정이었던 기계어 를 결정할수가 있읍니다.그러나 완성된 기게어는 2바이트로 할수가 있어서,첫번째 확보해 두었던 4바이트 중 2바이트가 남아 버립니다. 그래서 어셈블러는 불필요한 2바이트 에 NOP(=no operation:아무것도 하지 않는 것 에 상당하는 90H 를 삽입합니다.
이와 같이 첫번째읽었을 때에 아직 정의되어 있지 않은 라벨(상수 이름이나 변수이름 기타 등등)이 나타나는것을 전방참조(forward refenence) 라고 부릅니다.

전방참조 가 있으면 프로그램의 효율이 나쁘게 되는것뿐만 아니라 최악의 경우에는 어셈블 에러를 일으 킵니다.그래서 EQU 문에 의한 정의는 가능한한 프로그램의 선두에 놓도록 합니다.
물론 , DB ,DW 에 의한 데이터의 정의도 항상 전반참조의 문제를 가지고 있으모로 원래는 프로그램 선두에 놓는 쪽이 좋다 라고 말할수 있지마 EQU 문과는 달라서 단순히 선두에 가져다 놓으면 된다라고도 할수없으므로 ,이것에 대해서는 순서에 따라 다시 설명하겠읍니다.

```
MOV     DI,OFFSET ANS4
MOV     [DI],DX
MOV     [DI+2],AX
```

라고 함으로써 더블 워드는 변수로 2개의 워드형 데이터를 저장하고 있읍니다.

같은 수치의 곱셈이라도 부호가 달린 것으로 부느냐 안보느냐에 따라 결과가 다르다.

```
          MUL           >0FF0 ...   부호없는 곱셈인 경우
F0 * 11
          IMUL          >FEF0 ...    부호있는 곱셈인 경우
```

스트럭처(structure)에 의한 변수의 정의 :구조체라는 의미
데이터를 프로그램중에 나열하는 경우

이름\항목	AGE	HEIGHT	WEIGHT
A	45	170	72
B	38	164	58
C	25	175	68

```
A_AGE           DB      45
A_HEIGHT        DB      170
A_WEIGHT        DB      72
B_AGE           DB      38
B_HWIGHT        DB      164
B_WEIGHT        DB      58
C_AGE           DB      25
C_HWIGHT        DB      175
C_WEIGHT        DB      68
```

이와같은 방법에서의 정의는 ,데이터의 수가 증가됨에 따라 대단히 귀찮게 되고 또한 알기 어렵게 됩니다.

MASM에서는 몇가지 항목으로 나누어지는 것과 같은 데이터 구조를 정의 할수있도록 스트럭쳐형 변수 라고 하는 구조를 정의할수있도록 되어있음니다.
스트럭처형 변수를 사용하기 위해서는 미리 **STRUC 의사명령** 을 사용하여 데이터의 구조를 정의해 둡니다.

```
AHW     STRUC
AGE     DB      ?       스트럭처형 변수선언을 위한 의사명령 예
HEIGHT  DB      ?
WEIGHT  DB      ?
AHW     ENDS
```

여기에서 AHW가 스트럭처 이름이 되고 AGE,HEIGHT,WEIGHT가 **필드이름** 입니다.여기에서 정의된 스트럭처 이름은 데이터의 구조에 대해 붙여진 이름으로서 변수이름 그자체는 아닙니다.

```
A       AHW     <45,170,72>
```

와 같은 식으로 합니다.그리하여 A는 스트럭처 변수이름< > 속이 그 변수의 각 항목에 대해 주어진 데이터의 초기치 입니다.여러 사람의 데이터를 정의하려면 다음과 같이 합니다.

```
A       AHW     <45,170,72>
B       AHW     <38,164,58>
C       AHW     <25,175,68>
```

이것으로써 A,B,C ... 이 3 개의 필드 (AGE,HEIGHT,WEIGHT) 를 가지는 스트럭처형 변수로서 정의되어 각 필도에 대응 하는 데이터가 설정됩니다.
스트럭처 이름(이경우는 AHW)은 단순히 데이터의 구조를 나타내는 것이라는 점에 주의해 주십시요.
스트럭처형 변수 내의 각각의 요소를 꺼내려면 ,스트럭처형 변수이름 다음에 피리오드(.) 와 필드 이름을 붙인 것을 사용합니다.

```
MOV     AL,A.AGE
MOV     BL,A.HEIGHT
MOV     CL,A.WEIGHT
```

라고 함으로써 AL 레지스터에 A 씨의 연령(=45),BL 레지스터에 A씨의 신장(=170), CL레지스터에 A 씨의 체중(=72)이 전송됩니다.
또한 전원의 연령의 합계를 구하려면

```
MOV     AL,A.AGE
ADD     AL,B.AGE
ADD     AL,C.AGE
```

라고 하면 되는것입니다.(오버 플로우처리(over flow) 는 생략)

스트럭처의 사용순서는

스트럭처 이름에 대한 데이터 구조의 정의

\/

변수이름에 대하여 스트럭처 이름을 사용하여 스트럭처형
변수라는 것을 정의함과 동시에 초기화 데이터를 준다.

단순히 스트럭처 선언을 한것만으로는 메모리 상에 데이터는 설정되지않습니다.
각필드 이름이 가지는 스터럭처 선두로 부터의 오프셋 번지()가 표시되어있을 뿐
실제 번지상에 데이터가 세트되는 것은 아닙니다.
A AHW < , , > 에 의하여 데이터가 할당됩니다.

*.디버거의 G 커멘드는 코드세그머트에 대하여
D 커멘드는 데이터 세그먼트를 기초로 하여 오프셋번지를 상용하도록
되어있습니다.

예제) MUL3.ASM
벡터 의 내적계산

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
        MOV     AX,CODE       DS설정
        MOV     DS,AX
        MOV     AL,VCTA.X
        IMUL    VCTB.X
        MOV     BX,AX
        MOV     AL,VCTA.Y
        IMUL    VCTB.Y
        ADD     BX,AX
        MOV     AL,VCTA.Z
        IMUL    VCTB.Z
        ADD     BX,AX
        MOV     ANS.BX
;
        MOV     AH,4CH
        INT     21H
;
ANS     DW      ?
;
VCT     STRUC
X       DB      ?       ; X,Y,Z는 필드이름
Y       DB      ?
Z       DB      ?
VCT     ENDS
;
VCTA    VCT     <1,-2,3>

VCTB    VCT     <2,3,-1>
```

```
;
CODE    ENDS
        END
```

5.11 나눗셈 명령

나눗셈명령에는 부호없는 나눗셈을 하는 DIV(divide)명ㄹ령과 ,부호가 붙은 나눗셈을 하는 IDIV(Integer divide)명령등이 있읍니다.나눗셈은 기본적으로는 곱셈명령의 역함수이므로 곱셈명령으로 부터 어느정도 사용법등을 생각해 낼수가 있읍니다.
정수끼리 나눗셈을 하였을때,일반적으로 깨끗히 나눠져 결과가 얻어진다.고 말할수없읍니다.그래서 8086CPU에서는 나눗셈의 결과는 몫과 나머지로 나누어 2개의 레지스터에 저장합니다.

16비트 % 8비트

피젯수		젯수		나머지	몫
AX	%	8비트의 레지스터혹은 메모리	>	AH	AL

32비트 % 16비트

피젯수		제수		나머지	몫
DX AX	%	16비트의 레지스터혹은 메모리	>	DX	AX

이경우 의 DX:AX레지스터 표현은 지금까지와 마찬가지로 32비트 레지스터로서 사용되고 있읍니다.결과가 몫과 나머지라는 형태로 된다는 것 외에는 ,곱셈의 역연산(逆演算) 이라고 생각하면 외우기 쉬우리라고 생각됩니다.
여기서 주의하지않으면 안되는 것은 부호가 있는 나눗셈의 경우로서 ,음수를 포함하는 나눗셈일 경우의 나머지 부호는 어떻게 할것인가 라는 사항인데 ,8086CPU에서 는 피젯수의 부호와 나머지의 부호가 일치하는 결과를 구하도록 되어있읍니다.
사용법:

```
DIV    범용 레지스터 (8/16 bit)
       메모리        (8/16 bit)
IDIV    범용레지스터 (8/16 bit)
        메모리       (8/16 bit)
```

명령의 사용법은 곱셈명령과 비슷해서 피젯수를 미리 AX 레지스터 혹은 DX:AX 레지스터에 저장한다음 에 오퍼랜드를 젯수가 들어간 레지스터 혹은 메모리에 지정합니다.
16비트 % 8비트 인지 ,32비트 % 16비트인지는 젯수의 지정에 의한 레지스터 혹은 메모리의 크기에 따라 결정됩니다.

0 에 의한 나눗셈의 처리 :
여기서 문제되는 것은 ,0에 의한 나누셈과 오버플로우(overflow,자리넘침)인 경우입니다. 나눗셈에 젯수로서 0 을 지정한 경우 ,나눗셈을 할수가 없읍니다.이와같은 경우에는 명령실행을 중단하고 INT 0 인터럽트 를 발생하여 0 에 의한 나눗셈 처리 루틴으로 실행을 옮깁니다. [인터럽트란 일종의 서브루틴 호출을 말합니다.]
INT 0 인터럽트는 0 에 의한 나눗셈을 실행할때에 CPU 가 자동적으로 개입하여 실행하는 특수한 인터럽트입니다.이루틴 중에 0 으로 나눈 경우의 처리법을 미리 작성해 둡니다.(보통은 O.S. 등이 설정해 둔다.)
또한 나눗셈을 실행할때에 오버플로우가 발생하는 경우가 있읍니다.이것은 나눗셈의 결과를 저장하는 레지스터의 크기가 정해져 있기 때문에 ,예를 들면 16비트%8비트에 있어서 FFFF%1

> FFFF

로된경우,결과가 8비트로 된 AL 레지스터에 들어가지 않으므로 오버플로우가 발생합니다.이와 같은 경우도 8086에서는 INT 0 를 발생하여 실행을 중단합니다.

예제) DIV1.ASM
삼각형의 중심을 구하는 문제(나눗셈 설명용)

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
        MOV     AX,DATA      DS 설정
        MOV     DS,AX
;
        MOV     AL,A.X
        CBW                  ;AL을 AX로 부호확장
        MOV     BX,AX
        MOV     AL,B.X
        CBW
        ADD     BX,AX
        MOV     AL,C.X
        CBW
        ADD     AX,BX
        MOV     BH,3
        IDIV    BH    ; 3으로 나눈다
        MOV     M.X,AL
;
        MOV     AL,A.Y
        CBW
        MOV     BX,AX
        MOV     AL,B.Y
        CBW
        ADD     BX,AX
        MOV     AL,C.Y
        CBW
        ADD     BX,AX
        MOV     AL,C.Y
        CBW
        ADD     AX,BX
        MOV     BH,3
        IDIV    BH
        MOV     M.Y,AL
;
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
;-----------------
DATA    SEGMENT
P       STRUC
X       DB      ?
Y       DB      ?
P       ENDS
```

```
;
A       P       <12,41>
B       P       <-53,-19>
C       P       <?,?>
;
DATA    ENDS
        END
```

예제)DIV2.ASM
초를 시간 ,분,초로 바꾸는 문제

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
SIXTY   EQU     60
;
        MOV     AX,DATA
        MOV     DS,AX
;
        MOV     SI,OFFSET SECOND
        MOV     AX,[SI]
        MOV     DX,[SI+2]
        MOV     BX,SIXTY
        DIV     BX
        MOV     TIME2.SEC,DL
        MOV     BL,SIXTY
        DIV     BL
        MOV     TIME2.MIN,AH
        MOV     TIME2.HOUR,AL
;
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
;------------------
DATA    SEGMENT
TIME    STRUC
HOUR    DB      ?
MIN     DB      ?
SEC     DB      ?
TIME    ENDS
;
SECOND  DD      72912
TIME2   TIME    < , , >

DATA    ENDS
        END
```

주의 해야할점은 ,DD 에 의해 정의된 숫자는 메모리상에서 는
1바이트씩 완전하게 순서로 나열된다는 것입니다.
72912(10진수)= 00 01 1C D0 (16진수)
메모리상의 데이터 D0 1C 01 00

하위16비트(역워드) < > 상위16비트(역워드)