6.5 프로시듀어의 배치

PROC ~ ENDP 는 사람이 보기 편하기 위한 구분에 불과하다. 어셈블러는 실제로 프로시듀어를 구별하지 않는다.

예제)CALL2.ASM

```
CODE    SEGMENT
        ASSUME  CS:CODE
EOF     EQU     1AH

START:  CALL    GETCH
        JC      EXITP   ;    여기에서 프로시듀어를 호출하고 있다.
        MOV     DL,AL
        CALL    PUTCH   ;
        JMP     START
EXITP:  MOV     AH,4CH
        INT     21H

GETCH   PROC
        MOV     AH,8
        INT     21H
        CMP     AL,EOF
        JNE     GETCHEND
        STC
        RET
GETCHEND:                   ;긴이름의 라벨도 사용가능 그때라벨만의 행으로 해도좋다
        CLC
        RET
GETCH   ENDP

PUTCH   PROC
        MOV     AH,2
        INT     21H
        RET
PUTCH   ENDP

CODE    ENDS
        END     START    ;실행시작을 알리는 문장
```

END 문이 가지는 기능: 프로그램의 시작번지를 지정한다.
이와같은것은 여러개의 코드 세그먼트를 가진 프로그램을 작성한다든지 ,프로시듀어와 메인 루틴의 순서를 바꾼 프로그램을 작성하는 경우에는 절대로 필요하다.

예제)CALL2-2.ASM

```
CODE    SEGMENT
        ASSUME  CS:CODE
EOF     EQU     1AH

GETCH   PROC
        MOV     AH,8
        INT     21H
        CMP     AL,EOF
        JNE     GETCHEND
```

```
        STC
        RET
GETCHEND:                   ;긴이름의 라벨도 사용가능 그때라벨만의 행으로 해도좋다
        CLC
        RET
GETCH   ENDP

PUTCH   PROC
        MOV     AH,2
        INT     21H
        RET
PUTCH   ENDP

START:  CALL    GETCH   ; 메인 루틴의 시작          <
        JC      EXITP   ;   여기에서 프로시듀어를 호출하고 있다.
        MOV     DL,AL
        CALL    PUTCH   ;
        JMP     START
EXITP:  MOV     AH,4CH
        INT     21H

CODE    ENDS
        END     START   ;실행시작을 알리는 문장      >
```

제7장 모듈별 프로그램의 작성법

7.1 INCLUDE 의 방법

모듈화의 이점: 프로그램을기능 별로 분활하여 알기쉽게 하는 것과 걔발효율을
향상시킬수있다
아무리 같은 루틴을 사용할수있다고 하더라도 ,다른 프로그램을 만들때마다. 같은 루틴을 키보드로 부터 입력하지 않으면 안된다는 것은 ,역시귀찮은 작업입니다.
그래서 완성된 루틴의 부분을 미리 다른 화일 로 부터 작성해두고,메인 루틴만을 에디터로 부터 입력하여,서브루틴인 부분은 나중에 화일 단위로 결합시키는 방법을 생각해 냈읍니다.이것이 어셈블러에 있는 인클루드(INCLUDE) 기능입니다.
이기능을 사용할 때에는 우선 ,서브루틴 등의 부분을 독립시킨 화일로서 작성해두고 , 메인 루틴을 작성할때에 그중 필요한 부분으로 INCLUDE 의사명령을 사용하여 "INCLUDE 화일 이름 " 으로 한다음 ,결합하는 화일 이름을 지정합니다.이렇게 해두면 메인 루틴을 어셈블할때에 자동적으로 필요한 화일이 결합되어 어셈블됩니다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
INCLUDE PUTAL.SUB       ; PUTAL.SUB 화일의 내용을 인용한다.
;
START:  MOV     AX,CODE
        MOV     DS,AX

        MOV     AL,3FH
        CALL    PUTAL

        MOV     AH,4CH
        INT     21H

CODE    ENDS
        END     START
```

B 드라이브에 PUTAL.SUB 화일이 들어 있을때
INCLUDE B:PUTAL.SUB

INCLUDE 로 전개된 부분의 어셈블된리스트 화일은 "C " 가 붙어있다.

동일 변수에 ,여러개의 데이터를 나열할때에는 다음 행에 변수이름을 가지지않는 DB(define byte) 선언 등을 계속한다.

```
        NUM     DB      12H,34H
```

는

```
        NUM     DB      12H
                DB      34H
```

라고 한것과 같다.

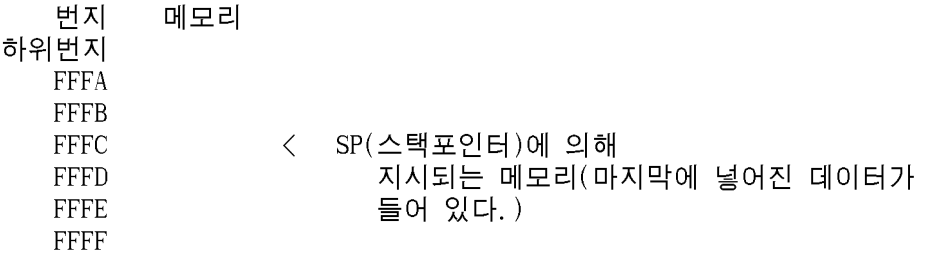
레지스터의 대피:

```
MOV     BXSAVE,BX
CALL    PUTAL           BX 레지스터의 값을 대피했다가 복귀한다.
MOV     BX,BXSAVE
        ..........
BXSAVE  DW      ?       ; 변수 선언
```

7.2 PUSH,POP 명령

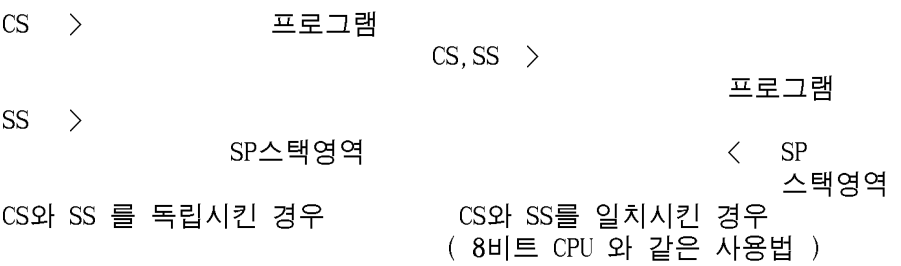
PUSH,POP 명령은 ,스택 영역이라고 부르는 특별한 영역에 대하여 데이터를 기록,읽어내는 명령입니다.

스택동작(stack operation)
마지막에 넣은 것을 처음에 꺼내는 동작(LIFO,Last In First Out)을 스택동작
메모리는 순서대로 나열되어 있으므로 ,현재어디까지 데이터가 쌓여있는지를 나타내기 위하여,스택 포인터(stack pointer,ST 레지스터의 일종)에 마지막에 놓여진 데이터가 있는 메모리의 번지를 기억하도록 되어있읍니다.
새로운 데이터를 쌓아올리면 SP 값은 감소 하고,데이터를 꺼내면 SP 값은 증가



주의) 보통의 경우와 ,데이터의 저장법과 순서가 거꾸로 되어 있으므로 주의가 요함
8086의 PUSH, POP 명령에서는 반드시 2 바이트의 데이터가 한번에 읽어들여지고 꺼내어지므로, SP 의 값은 항상 2씩 증감합니다.
스택영역은 보통 메모리의 최상위 번지로 설정되지만 이것은 8비트 CPU 와 같이 메모리공간이 좁은 시스템에 있어서 프로그램과 스택상의 데이터가 충돌하지 않게 하기위해서 였읍니다.
8086에서는 보다 넓은 메모리 공간을 사용할수 가 있으므로 ,스택 동작을 위해서 스택 세그먼트를 독립하여 설정하고 있읍니다.
SP 가 가리키는번지는 실제는 스택 세그먼트(SS)가 가리키는 베이스 번지로 부터의 오프셋 번지입니다.

스택 세그먼트의 설정법에 의해 프로그램이나 데이터와 스택롸를 독립하여 배치할수도 있고,또한 8지트 CPU와 같이 프로그램과 스택을 동일한 메모리 공간에 배치할 메모리 공간에 배치할수도 있읍니다.
어느경우에서도 스택 포인터의 초기치를 적당하게 설정함으로써 적당한 번지를 스택영역의 시작 번지로 할수가 있는데 ,보통은 최상위 번지로 부터시작합니다.



CS와 SS 를 독립시킨 경우 CS와 SS를 일치시킨 경우
(8비트 CPU 와 같은 사용법)

PUSH 16비트 범용 레지트터
세그먼트 레지스터
16비트 메모리
POP 16비트 범용 레지스터
세그먼트 레지스터
16비트 메모리

*.POP CS 명령만은 존재하지 않는다.
IP를 PUSH, POP 하는 명령은 없다.
플레그 레지스터를 PUSH, POP 하는 명령으로는 위와 별도로 PUSHF, POPF명령을 사용

PUSH, POP 명령의 실제 사용 예:
레지스터의 대피가 필요 하게 된것은 서브루틴 중에서 레지스터의 내용이 파괴 되기 때문입니다. 어느 레지스터의 내용이 파괴되는지는 서브루틴의 내용을 보지않고서는 알수없읍니다. 그래서 **레지스터의 대피, 복귀는 메인 루틴쪽이 아니라 서브루틴 쪽에서 하는 것이 적절 합니다.**

```
메인루틴                          메인루틴
........                            .....
PUSH BX                           CALL PUTAL
CALL  PUTAL     <                    .....
POP  BX
```

```
서브루틴                          서브루틴
                                  PUSH BX
PUSH ,POP 를 메인루틴쪽에서          .....
   행한경우의 프로그램            POP BX
```

PUSH,POP를 서브루틴쪽에서 행한 경우의 프로그램 완전히 같은 동작을 하지만 오른쪽이 더 좋은 프로그램입니다.
메인 루틴쪽에서 어느 레지스터가 서브루틴 내에서 변경이나 파괴되는가를 생각하지 않아도 되기 때문입니다.
서브루틴을 완전히 블랙 박스 로서 사용할수가 있기 때문입니다.
이와 같이 함으로써 서브루틴의 모듈화가 점점 효율적으로 됩니다.

예제)PUSH1.ASM
BL에 초기치로 주어진 어떤 8비트 값()을 1배,2배,..., F배 한수치를 출력한다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
INCLUDE PUTAL2.SUB
INCLUDE CRLF.SUB
;
ENTRY:  MOV     AX,CODE
        MOV     DS,AX       DS설정
;
        MOV     BL,SEED ;초기 데이터 21H를 BL 에 지정한다.
        MOV     CL,1
NEXT:   CMP     CL,10H  ;10H배이면 끝남.
        JE      EXITP
        MOV     AL,CL     초기 데이터의 BL배를 구한다.
        MUL     BL,
        PUSH    AX
        MOV     AL,AH             상위 바이트 출력
        CALL    PUTAL
        POP     AX              AX레지스터를 보존(출력에 AL 레지스터를 사용하기
```

```
        CALL    PUTAL   ;하위 바이트 출력                                  때문  )
        CALL    CRLF    ;개행
        INC     CL
        UMP     NEXT
EXITP:  MOV     AH,4CH
        INT     21H         끝내고MS-DOS 로 귀환
;
SEED    DB      21H  ;초기 데이터
CODE    ENDS
        END     ENTRY


PUTAL2.SUB
PUTAL:
        PUSH    AX    ;서브루틴 속에서 레지스터
        PUSH    BX     AX,BX,DX 를 대피하고 있다
        PUSH    DX
        MOV     BL,10H
        MUL     BL
        MOV     DL,AH
        CALL    PUTHEX  ; 상위자리를 출력
        MOV     AH,0
        MOV     BL,10H
        DIV     BL
        MOV     DL,AL
        CALL    PUTHEX  ;하위자리를 출력
        POP     DX     AX,BX,DX 를 복귀
        POP     BX      (PUSH와 의 역순)
        POP     AX
        RET
;
PUTHEX: PUSH    AX
        PUSH    DX
        CMP     DL,0AH
        JAE     HEX2
        ADD     DL,'0'
        JMP     HEX3
HEX2:   ADD     DL,'A'-0AH
HEX3:   MOV     AH,2
        INT     21H
        POP     DX
        POP     AX
        RET
```

```
CRLF.SUB
CRLF:
        PUSH    AX
        PUSH    DX
        MOV     DL,0DH  ;CR(복귀)을 출력
        MOV     AH,2
        INT     21H
        MOV     DL,0AH
        MOV     AH,2    ;LF(개행)을 출력
        INT     21H
        POP     DX
        POP     AX
        RET
```

PUSH,POP 명령은 16비트로 된 레지스터에 ,혹은 메모리에 대피,복귀밖에 하지 못하므로 ,서브루틴 내의 DL레지스터 밖에 사용하지 않는경우에도 DX레지스터로서 대피한다.

```
        PUSH    AX  ; AX의 내용을 BX에 전송
        POP     BX
```

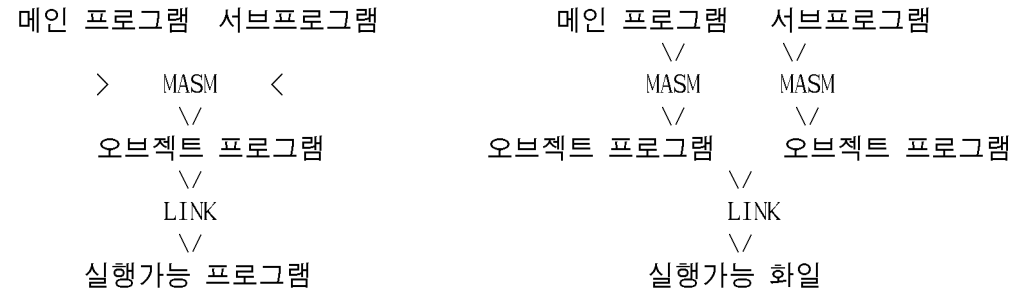
직접 전송을 허용하지 못하는 세그먼트간의 값을 전송하는데 유효합니다.

```
 예)    PUSH    CS
        POP     DS
```

가장주의 해야할점:
하나의 서브루틴 중에서 PUSH 한것은 그 루틴 내에서 반드시 POP 한다.

7.3 프로그램의 모듈화와 링클방법

모듈화된 프로그램을 따로 별도로 어셈블하고 ,LINK에 의해 결합 할수도 있읍니다.



어셈블단계에서 결합하는경우 링크 단계에서 결합하는 경우

오브젝트 단계에서 링크가 가능하기 때문에 ,MS-DOS에서느 공동형식 ㅇ의 오브젝트 화일을 작성하는 어셈블러나컴파일러의 출력을 합쳐서 하나의 실행가능 프로그램하는것이 가능하다.
이때문에 어셈블러와 C 언어와 의 링크 등이 가능하게 됩니다.
그러나 링크의 기능을 잘 사용하기 위해서는 ,많은 형식사의 약속이나 링커의 동작을 이해하지 않으면 안되아서 그 개요를 설명하는 것만해도 상당한 분량을 차지하게 됩니다.
프로그램의 개발에 대해서는 메인 프로그램을 포함하는 모듈의 소스 프로그램과 서브루틴으로 이루어지는 소스 프로그램으로 작성합니다.

예제)MODULE1.ASM
문자열(string) We are the world 를 출력한다.

```
CODE    SEGMENT PUBLIC
        ASSUME  CS:CODE,DS:DATA
;
EXTRN   PUTSTR:NEAR    외부에서 참조하는 서브루틴의 선언
;
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     BX,OFFSET MSG
        CALL    PUTSTR
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
;
DATA    SEGMENT PUBLIC
MSG     DB      'We are the world',00H
;
DATA    ENDS                              >끝내기 위해 삽입된 0 이다.
        END     START
```

PUTSTR.ASM
문자열을 출력하는 서브루틴

```
CODE    SEGMENT PUBLIC
        ASSUME  CS:CODE
;
PUBLIC  PUSTR   ;다른 모듈에서 참조되는 서브루틴 이름을 PUBLIC 의사명령으로
;                 선언한다. 즉 다른데서 PUTSTR을 참조할것이라고 선언함.
PUTSTR  PROC    NEAR
        PUSH    AX
        PUSH    DX
        PUSH    SI
        MOV     SI,0
PUT_NET:
        MOV     DL,[BX+SI]
        CMP     DL,0              >문자열중 0 이 나타나면 끝낸다.
        JZ      PUT_EXIT
        MOV     AH,2          반드시 서브루틴
        INT     21H            PROC ~   ENDP 로
        INC     SI             선언한다.
        JMP     PUT_NEXT
PUT_EXIT:
        POP     SI
        POP     DX
        POP     AX
        RET
PUTSTR  ENDP
;
CODE    ENDS   형식상 하나의 세그먼트에 있는 걸로한다.
        END
```

서브루틴 PUTSTR은 ,BX에 의해 간접지정되는 번지를 선두로 한데이터 열을
문자열로서 출력하고 ,0 이 나타나면 종료하는 것입니다.

PUBLIC 이라는 지정은 : **다른 모듈과 링크할 때에 같은 이름의 세그먼트가 있다면 그것을 합쳐서 하나로 하라**, 라는 지정입니다.

EXTRN(external) :
열거된 서브루틴이 외부의 모듈의 모듈 속에 들어있다는 것을 나타내고 있읍니다. 어셈블러는 더셈블 단계에서 같은 모듈 내의 서브루틴이나 라벨이 존재하지 않는 경우에는 Symbol not defined 에러를 발생하는데, EXTRN 의사명령에 의해 선언되어 있게되면 ,그서브루틴이나 라벨은 다른 모듈에 있어서 링크할때 결합할수있다고 간주되어 에러를 발생하지 않고 어셈블을 계속합니다.
라벨이름이 동일세그먼트 내에 배치되는 것인 경우에는 ,
: 에 바로 이어서 NER을 붙이고, 라벨의 속성을 선언하지 않으면 안됩니다.
만일동일 세그먼트내에 존재하지 않을 때에는 ,
: FAR 을 붙이지 않으면 안됩니다.

서브루틴 :
서브루틴에 놓이는 모듈도 하나의 세그먼트로서 형식을 갖추기위해 그 앞뒤를 SEGMENT ~ ENDS 로 둘러 쌉니다.여기 에서 세그먼트선언은 메인루틴과 같지않으면 안됩니다.(동일한세그먼트 이름이어야 한다) 데이터 세그먼트를 사용하지않을 때에는 ASSUME 문장으로 DS 선언을 하지 않도록 합니다.

PUBLIC :
의사명령 으로 PUTSTR 이라는 이름의 서브루틴(라벨 이름)이 다른 외부의 모듈에서 참조 가능하다는 것을 선언하고 있읍니다.이것은 이 루틴 중 어느 라벨 이름니 외부참조가 가능 하다는 것을 나타내는 것 입니다.
여기서 사용하고 있느 PUBLC 과 세그먼트 선언중에 사용되고 있느 PUBLC과는 다른 것입니다. 그리고 서브루틴 전후를 PROC ~ ENDP 에 의해 둘러싸고, 구조를 명확하게 합니다. 이전과 마찬가지로 단순히 라벨 이름만을 서브루틴으로 할수도 있지만, 별로 추천할만 하지하지는 않습니다. 그리고 PROC 의 다음의 NEAR 은 없어도 상관이 없지만,이것도 동일한 세그먼트 내에서 참조 되는 라벨이라는 것을 나타내고 있읍니다.
이상의 것을 정리하여 프로그램에 필요한 구조를 나타내고면 다음과 같이 됩니다.
(세그먼트 이름을 CODE , 서브루틴 이름을 PUTSTR 로 한다.)

```
          메인루틴
CODE    SEGMENT PUBLC
        ASSUME  CS:CODE.......
EXTRN   PUTSTR:NEAR
START:

          본 문

CODE    ENDS

          만일 있다면 데이터 세그먼트 등

        END     START
```

```
   서브루틴
CODE    SEGMENT PUBLC
        ASSUME  CS:CODE.....
PUBLC   PUTSTR
PUTSTR  PROC    NEAR

        본  문

PUTSTR  ENDP
CODE    ENDS
        END
```

오브젝트 화일을 링크할 때에는 화일 이름을 " + " 로 연결하여 지정합니다.
반드시 메인 루틴을 포함하는 모듈의 화일 이름을 선두에 지정합니다.

```
A>LINK MODULE1+PUTSTR,,MODULE1;
```

이것으로 실행가능 화일이 완성됩니다.
자기가 모듈을 만들어 나갈때에는 위례서 설명한 방법에 따라서 모듈으 개발하면 되는데, 아른 참고서 로 부터 인용할 경우에 반드시 위와 같은 대로 되어 있지 않은 경우가 있읍니다. 그와 같은 경우에는 세그먼트 이름을 고쳐 쓰는 방법 등으로 대처해 주십시오. 그래도 안되는겨우에는 원인을 여러가지로 생각 할수가 있지만,자세하게는 나중에 다루도록 하고 ,할수없이 서브루틴 자체를 맞도록 자기가 다시 베열 하여 고쳐 작성하는 등 시행착오를 해 보십시요.