제 8 장 . 반복 기법

8.1 루프(loop)명령

비교와 분기를 하나로 합쳐서 한명령으로 할수 있도록 된 루프(loop),스트링(string)

LOOP 명령
LOOP 명령은 CX레지스터를 뺄셈식 카운터로서 사용하고 ,CX 레지스터가 0 이 될때까지 어떤 문장들을 되풀이 반복한 뒤 지정한 라벨로 분기 하는 명령입니다.즉, CX 에 설정한 횟수만큼 반복을 하기위한 명령인것 입니다.
LOOP명령은 오퍼랜드에 분기하는 곳의 라벨을 지정하여,

```
        LOOP    NEXT
```

와 같은 식으로 사용합니다.이명령은 지금까지 알고 있는 명령을 조합하여 실행하려고 하면

```
        DEC     CX
        CMP     CX,0
        JNE     NEXT
```

와 같이 하면 됩니다.즉 , LOOP 명령은 CX 레지스터 뺄셈명령과 비교명령과 조건분기 명령을 한 명령으로 실행하는 명령입니다. 이경우 뺄셈 카운터로서 사용되는 것은 CX레지스터 라고 정해져 있읍니다.루프명령을 실행하면 CX 레지스터 값을 하나 빼고, 0 이아니면 지정된 라벨로 분기 합니다.
LOOP명령에 의해 CX레지스터에 지정된 횟수만큼 반복할수가 있읍니다.
CX 레지스터에는 LOOP명령에서 지정하는 반복범위보다 앞서서 반복하는 횟수를 설정해 주지 않으면 안됩니다.

예제)LOOP1.ASM
1부터 10H까지의 수치를 화면에 출력하는 프로그램

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
INCLUDE PUTAL2.SUB   - INCLUDE화일의 전개부분의 리스트는 생략하였다.
;
START:  PUSH    CS      DS 에 CS의 값을 지정함
        POP     DX
;
        MOV     AX,1
        MOV     CX,10H
L1:     MOV     DS,AL
        CALL    PUTAL
        CALL    SPACE       이범위를 10H회 반복한다.
        INC     AX
        LOOP    L1
        MOV     AH,4CH
        INT     21H
;
SPACE   PROC    NEAR    - NEAR지정은 없어도 좋다.
        PUSH    AX          레지스터를 퇴피
        PUSH    DX
```

```
        MOV     DL,' '   공백문자를 출력
        MOV     AH,2
        INT     21H
        POP     DX       퇴피했던 레지스터값을 회복
        POP     AX
        RET
SPACE   ENDP
;
CODE    ENDS
        END     START
```

위의 프로그램은 리스트는 INCLUDE 화일 이 전개된 부분은 생략하고 나타낸것 입니다
CX레지스터의 초기 설정은 루프밖에서 세트
LOOP명령은 ,그자신 속에 CX의 값을 빼는 명령을 포함하고 있다
LOOP : 무조건 지정된 횟수를 반복
LOOPE(loop while equal) ,LOOPNE(loop while not equal):
어떤 조건이 지정된 횟수 이내에 서 반복을 끝냄
이 명령 앞에 비교명령(또는 연산명령)을 놓고 그 결과가 같고 ,또한 CX 레지스터의 1 씩을 값을 뺀 결과가 0 이 아니면 반복하는 명령입니다.
상한을 CX 회로 설정해 놓고 같지않은 것이 나타날 때까지 반복하는 명령입니다.
이것은 데이터의 열중에서 서로 다른 데이터를 찾는데에 사용할수 있읍니다.

예제)LOOP2.ASM
 문자열 "_ _ _ _ _HELLO!" 의 공백의 갯수를 세고,문자열 "HELLO!_ _ _"중의 연속된 문자갯수를 세어 출력한다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
INCLUDE PUTAL2.SUB    INCLUDE화일의 전개부분은 생략한다.
INCLUDE CRLF.SUB
;
SPACE   EQU     ' '
;
START:  MOV     AX,DATA
        MOV     DS,AX
;
;COUNT  SPACE   LENGTH          공백길이를 센다.
        MOV     CX,10H         상한 회수를 10H회로 결정한다.
        MOV     DL,0           공백수를 DL레지스터에카운트
        MOV     SI,0           선두번지부터 몇번째인가를 나타내는 SI
        MOV     BX,OFFSET STR1
NEXT1:  INC     DL
        INC     SI              SI 가 INC SI 에 의해서 1부터 시작되므로
        MOV     AL,[BX+SI-1]    하나를 빼야한다.
        CMP     AL,SPACE        CX내용에서 1을 뺀 결과 0이아닌 동안 여기서 AL과
        LOOPE   NEXT1           SPACE와 비교한 결과 같으면 NEXT1으로분기한다.
        DEC     DL              같지않으면 아래로 계속한다.
        MOV     AL,DL           카운트에 더 더한 만큼 뺀다
        CALL    PUTAL           표시
        CALL    CRLF            줄을 바꾼다.
;
```

```
;COUNT STRING LENGTH                    문자열겟수를 센다
        MOV     CX,10H
        MOV     DL,-1          초기값을 -1로 하는것은
        MOV     SI,-1          0FFFFH세트하는 것
NEXT2:  INC     DL
        INC     SI
        MOV     AL,[BX+SI]
        CMP     AL,SPACE       CX를 1감소시켰을 때 0이 아닌범위 내에서 AL와
        LOOPNE  NEXT2          SPACE가 같지 않으면 NEXT2로 분기하여 문자 갯수
        MOV     AL,DL          를 센다.
        CALL    PUTAL          표시
        CALL    CRLF           줄을 바꾼다.
;
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
DATA    SEGMENT
STR1    DB              HELLO!' ;선두가 공백인 문자
STR2    DB      'HELLOW!   '    ;선두부터 문자가 들어있는 문자
DATA    ENDS
        END     START
```

LOOP 명령과 플레그 레지스터:
LOOP 명령, LOOPE명령,LOOPNE명령은 모두 CX 레지스터를 카운터로 하고 한번 실행할때마다 CX 레지스터 값을 하나 씩 값을 빼고 있읍니다.그러나 여기에서 CX 레지스터 값을 뺄떼는 ,결과가 무엇이든 상관없이 플래그 레지스터의 값은 변화하지 않습니다.
즉, LOOP명령등의 실행 후 플래그 레지스터의 값은 앞의 연산결과를 유지하고 있읍니다.따라서 LOOP 명령 중의 CX 레지스터를 빼는 명령은 단순한 DEC 명령과 치환할수는 없읍니다.(DEC 명령에서는 CF이외의 레지스터가 변화한다.). LOOP명령,LOOPE명령,LOOPNE 명령을 이해를 돕기 위해서 다른 명령으로 치환 한다면 다음과같이 됩니다.

LOOP명령 LOOPE명령, LOOPNE명령들의 똑같은 기능을 다른 명령으로 치환

```
        LOOP L1          LOOPE L1         LOOPNE L1

        PUSHF            PUSHF            PUSHF
        DEC   CX         JNZ   EXIT       JZ    EXIT
        JE    EXIT       DEC   CX         DEC   CX
                         JZ    EXIT       JZ    EXIT
        POPF             POPF             POPF
        JMP   L1         JMP   L1         JMP   L1
   EXIT:POPF        EXIT:POPF        EXIT:POPF
```

여기서 PUSHF 는 플래그 레지스터(flag register)를 스택에 퇴피시킵니다.

8.2 스트링(string) 명령

연속한 데이터를 전송하는 명령(5종류)
문자열과 같은 연속한 데이터를 전송하는 명령입니다.
리피트 프리픽스(repeat prefix) 명령과 조합하여 사용함으로써 ,한명령으로 대량의 데이터를 전송할수 있는 이점이 있읍니다.

스트링 명령

		바이트 단위		워드 단위	
전송 명령군	LODSB	메모리	>AL	LODSW	메모리 > AX
	STOSB	AL	>메모리	STOSW	AX > 메모리
	MOVSB	메모리	>메모리	MOVSW	메모리 > 메모리
비교 명령군	SCASB	메모리 AL의 비교		SCASW	메모리와 AX의 비교
	CMPSB	메모리끼리의 비교		CMPSW	메모리와 메모리의 비교

리피트 프리픽스 명령

전송 명령군	REP	CX 레지스터에 주어진 횟만큼,다음의 스트링 명령을 반복
비교 명령군	REPE REPZ	비교결과가 0 및 CX레지스터에 주어진 횟수 이내이면 다음의 스트링 명령을 반복한다.
	PEPNE PEPNZ	비교결과가 0이 아니고 CX레지스터에 주어진 횟수 이내이내 이면 다음의 스트링 명령을 반복한다.

스트링 명령에서는 메모리의 번지를 지정하는 방법이
정해져 있고,또한 데이터를 읽는 방법과 쓰는 방법이 차이가 있읍니다.
*.데이터를 읽어낼때...DS 내의 [SI]로 표시되는 번지의 내용을 읽어낸다.
*.데이터를 써넣을때...ES 내의 [DI]로 표시되는 번지의 데이터를 써넣는다.

사용하는 번지도 바이트 단위면 AL, 워드 단위이면 AX 로정해져 있읍니다.따라서 스트링 명령을 사용하기 전에는 필요한 레지스터(DS,ES,SI,DI,CX)의 초기설정을 하지않으면 안되는 대신에,명령자체는 오퍼랜드를 가지지 않고 단순히
LODSB
MOVSW
와 같은 식으로 단순하게 사용할수가 있읍니다.

단,스트링 명령은 전송이나 비교 등을 행한 다음,번지지정에 사용되어 SI혹은 DI레지스터의 값을 자동적으로 1(바이트 단위일때) 또는 2(워드 단위일때) 만큼 증가시킵니다. 이것은 다음번지의 데이터를 전송 또는 비교하기위한 준비이고,스트링 명령을 연속하여 사용 함으로써 연속된 번지에 놓여진 데이터를 전송 또는 비교할수있읍니다.
(디렉션 플레그값이 1 일때는 ,SI, DI 갑은 감소 된다.)

LOOP 명령을 사용하기 위해,편의상 미리 CX 에 문자열의 길이를 넣어두지 않으면 안됩니다. 문자열의 길이를 계산하기위하여 원하는 문자열의 다음에 더미(dummy) 변수 MSG2를 놓아서 두 개의 오프셋 번지의 차를 취합니다.
OFFSET MSG2-OFFSET MSG

이와 같이 매크로 어셈블에서는 번지등의 덧셈 뺄셈을 소스 프로그램 중에 쓸수가 있읍니다. 이 방법은 정석으로 되어 있읍니다.

LOODSB명령은 ,데이터를 전송한다음 SI 값을 하나 증가 시키므로, 한 문자 출력한 다음 LOOP명령으로 돌아와서 다시 LODSB 명령을 실행하게 되면, SI 는 앞에 출력한 문자가 있는 번지의 다음 번지를 나타내고 있읍니다.(연속문자 출력가능)

STOS명령
메모리로 부터 메모리로 데이터 전송
LODS 와 STOS를 조합하는 방법 ,MOVS 를 사용하는 방법,

예제)MOVS2.ASM
MSG1에들어 있는 스트링 FUNCTION 09H:DISPLAY STRING 을
MSG2로 전송하는 펑션 9 를 호출하여 출력한다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:DATA
;
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     ES,AX
;
        MOV     CX,OFFSET MSG2-OFFSET MSG1
        MOV     SI,OFFSET MSG1
        MOV     DI,OFFSET MSG2
        CLD      ;DI의 증가 설정
NEXT:   LODSB    ;DS:[SI]로 정해지는 번지메모리 내용을 ES:[DI]로
        STOSB    ;정해지는 메모리에 전송한다.단,문자열의 길이(CX) 만큼
        LOOP    NEXT
        MOV     BYTE PTR ES:[DI],'$'
        MOV     DX,OFFSET MSG2
        MOV     AH,9H
        INT     21H
;
CODE    ENDS

DATA    SEGMENT
MSG1    DB      'FUNCTION 09H:DISPLAY STRING'
MSG2    DB      20H DUP (?)
DATA    ENDS
        END     START
```

LODSB의 실행후 SI의 값이 하나 증가하고,STOSB의 실행후 DI 의 값이 하나 증가,
2 개의 명령에 LOOP 명령을 더함으로써 연속한 데이터를 전송할수가 있읍니다.

MOVS 명령:
위의 LODS 명령과 STOS 명령을 MOVS 명령으로 치환

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:EXTRA
;
START:  MOV     AX,DATA
        MOV     DS,AX       ;DS설정
        MOV     AX,EXTRA    ;독립된 ES의 설정이 필요
        MOV     ES,AX
;
        CLD                 ;DI의 증가 설정
        MOV     DX,5        ;5번 되풀이
        MOV     DI,OFFSET MSG2
NEXT2:  MOV     CX,OFFSET DMSG - OFFSET MSG1
        MOV     SI,OFFSET MSG1
NEXT:   MOVSB               ;CX의 값이 0이 될때 까지 MSG1번지부터 시작하는 스트링
        LOOP    NEXT        ;을 MSG2 번지위치부터 옮겨 놓는다.
        DEC     DX          ;DX 에서 1을 빼라
        JNZ     NEXT2       ;0 이 아니면 NEXT2로 가라.
        MOV     BYTE PTR ES:[DI],'$'   ;메모리에 $(문자열 끝표시)를 넣는다.
        MOV     AX,ES
        MOV     DS,AX       ;펑션호출 9번의 앞에는 DS의 설정이 필요
        MOV     DX,OFFSET MSG2
        MOV     AH,9H       ;DX에 지정된 번지부터 저장된 스트링을 $ 자가 나타
        INT     21H         ;날때까지 출력하는 시스템 펑션 호출.
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
;
DATA    SEGMENT
MSG1    DB      'REPEAT SAME MESSAGE 5 TIMES!'
DMSG    DB      ?
DATA    ENDS
;
EXTRA   DB      100H DUP (?)
EXTRA   ENDS
        END     START
```

8.3 리피트 프리픽스(1)

리피트 프리픽스 명령은 스트링 명령과 조합하여 사용하고 ,스트링 명령을 일정 횟수 연속하여 사용 하는것 입니다.
REP　　MOVSB

　　>이것이 리피트 프리픽스이다.
　　CX 에 지정된 횟수만큼 MOVSB를 되풀이 하라는 의사명령
　　리피트 프리픽스는 스트링 명령 앞에 놓고,스트링 명령의 기능 확장을 지정

여기서 STOSB　>B는 바이트를 나타냄
　　STOSW　>W는 워드를 나타냄
따라서 AX 에　AB 가 들어 있어도　STOSB는　B 만을 나타낸다.
AX 레지스터에　AB'를 대입하고 있으므로, AL 레지스터의 내용은 'B'=42H가 됩니다.
워드단위의 스트링 명령을 사용한 경우에는 20H 개의 바이트 의 데이터가 전송됩니다.
이것을 실행하면
A>MOVS5
BABABABABABABABABABA
AX레지스터의 값　AB' 가 메모리 상에 전송돌 경우에는 역워드(reverse word)형식으로 되기 때문에 ,'BA' 순서로 되어버린 점에 주의 하십시요.

REP 명령은 조합하는 스트링 명령이 바이트 형인가 워드 형인가에 따라서 전송되는 바이트 수는 달라집니다.
예를들면 20H 바이트의 데이터를 전송할때에 MOVSB명령을 사용ㄹ한다고 하면 CX 레지스터의 초기치를 20H로 ,MOVSW명령을 사용한다고 하면 CX 레지스터의 초기치를 10H 로하지 않으면 안될것입니다.
전송하는 데이터의 수가 홀수 바이트 일때에는 워드형의 스트링명령을 사용하면 마지막 데이터가 전송되니 않는다든지 ,여분의 데이터 까지 전송 된다든지 하는 일이 있으므로 주의가 필요합니다.　미리 데이터수를 알과있는 경우는 괜찬지만 경우에따라 전송하는 데이터수가 변화 할때에는 바이트 형의 스트링 명령을 사용하는 쪽이 안전 합니다.

8.4 비교명령군의 사용법

메모리와 레지스터의 값을 비교하는　SCAS명령,
메모리끼리의 값을 비교하는　　CMPS명령

SCAS명령
AL 레지스터의 값과 ES:[DI]에 의해 표시되는 메모리의 내용을 비교

몇번 반복했는가를 알기위해서 ,
원래의 CX값을 보존해두어,루프에서 빠져나온 후의 CX 값을 빼면 된다.
문자열길이의 계산 : STR2 - STR1
　　OFFSET STR2 - OFFSET STR1
과 완전히 같은 역활을 합니다. 이것도 하나의 정석으로서 외워두십시요

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE,ES:CODE
;
INCLUDE PUTAL2.SUB
START:  MOV     AX,CODE  ;
        MOV     DS,AX
        MOV     ES,AX    ;DS와 ES 의 설정
        CLD              ;DI 의 증가 설정
        MOV     AL,'&'   ;문자열 중에서 찾는문자
        MOV     DI,OFFSET STR1
        MOV     CX,STR2-STR1    ; 문자열의 길이를 계산하려면
        PUSH    CX              ;OFFSET 을 생략해도 좋다.
NEXT:   SCASB                   일치하는 문자가 있는가 찾는다.
        JE      FOUND           (AL의값과 ES:[DI]로 지정된 메모리 내용과 내용을 비교한
        LOOP    NEXT             다,그리고 자동적으로 DI는 증가 CX 는 감소한다.)
        JMP     NOTFOUND
FOUND:  DEC     DI
        MOV     DX,OFFSET FOUNDMSG1
        MOV     AH,9
        INT     21H
        MOV     AX,DI            발견한 번지를 표시
        MOV     AL,AH
        CALL    PUTAL
        MOV     AX,DI
        CALL    PUTAL
        MOV     DX,OFFSET FOUNDMSG2
        MOV     AH,9
        INT     21H
        POP     AX              문자열의 길이-뺄셈 카운터에 의해 몇번째 문자에서
        SUB     AX,CX            발견되었는지를 계산하여 표시
        CALL    PUTAL
        JMP     EXITP
NOTFOUND:
        MOV     DX,OFFSET NOTFOUNDMSG
        MOV     AH,9
        INT     21H
EXITP:  MOV     AH,4CH
        INT     21H
;
FOUNDMSG1       DB      'FOUND ADDRESS = $'
FOUNDMSG2       DB      'H COUND = $'
NOTFOUNDMSG     DB      'NOT FOUND $'
STR1    DB      'ABCDEFG&ABC'
STR2    DB      ?
CODE    ENDS
        END     STRAT
```

CMPS 명령 :
2개의 데이터열이 같은가 다른가를 비교하는데 사용합니다.
DS:[SI]가 지정하는 메모리 한 바이트 내용과
ES:[DI]가 지정하는 메모리 한 바이트 내용을 비교하는 명령이다.
CMPSB명령을 하면 SI,DI의값이 자동적으로 하나만큼 증가 한다는 점입니다.

예제)CMPS2.ASM
메모리 번지 CMP-STRING 부터 저장되어 있는 ABC" 문자와 키보드로 부터 입력하여 메모리 번지 BUFF부터 저장된 문자 스트링 속에 같은 "ABC" 가 몇 번째 글자부터 있는가 알아 낸다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE,ES:CODE
;
INCLUDE CRLF.SUB
INCLUDE PUTAL2.SUB
START:  MOV     AX,CS
        MOV     DS,AX
        MOV     ES,AX
        CLD
        MOV     DX,OFFSET OPENING_MSG
        CALL    DISP
        CALL    CRLF
        MOV     DX,OFFSET MAX_CHARS     리터키가 눌릴때까지 키보드로부터 입력된
        MOV     AH,0AH                  스트링을 연속적으로 BUFF속에 저장하고 총
        INT     21H                     입력된 문자갯수를 CHARS-ENTERED에 지정하
        CALL    CRLF                    는 시스템 펑션 A 번이다.
        MOV     CH,0
        MOV     CL,CHARS_ENTERED
        MOV     BP,OFFSET BUFF
        DEC     BP
        INC     CX
NEXT:   DEC     CX
        CMP     CX,3
        JB      NOTFOUND
        INC     BP
        MOV     DI,BP
        MOV     SI,OFFSET CMP_STRINGS
        CMPSB       <   DS:[SI]로 지정되는 메모리내용과  ES:[DI] 로 지정되는
        JNE     NEXT    메모리 내용을 비교한다.비교한후,자동적으로 DI,SI값
        CMPSB       <   을 증가시켜 다음것을 비교한다.
        JNE     NEXT
        CMPSB       <
        JNE     NEXT
        MOV     DX,OFFSET FOUND_MSG1
        CALL    DISP                    몇번째 문자에서 발견되었는지를계산표시
        MOV     AL,CHARS_ENTERED
        SUB     AL,CL
        CALL    PUTAL
        JMP     EXITP
NOTFOUND:
        MOV     DX,OFFSET NOTFOUND_MSG
        CALL    DISP
EXITP:  MOV     AH,4CH
        INT     21H
DISP    PROC
        MOV     AH,09H
        INT     21H
```

```
        RET
DISP    ENDP
OPENING_MSG     DB      'INPUP STRINGS INCLUDE'
<ABC> $'
FOUND_MSG1      DB      'FOUND CHARACTERS AT $'
NOTFOUND_MSG    DB      'NOT FOUND $'

MAX_CHARS       DB      80
CHARS_ENTERED   DB      ?
BUFF            DB      80H DUP (0)
```

>최대 입력가능 문자 갯수(CR포함):1바이트

버퍼 >입력된 실지 문자가 들어가는곳

>실제로 입력된 문자갯수(CR포함하지 않음) : 1바이트

```
CMP_STRINGS     DB      'ABC'
```

>비교대상이되는 문자열

```
CODE    ENDS
        END     START
```

위문제는 상당히 논리가 어려운 문제입니다.그러나 꾸준히 추적해나가면 곧 이해할수 있을 것입니다.그러나 이 프로그램을 완전히 이해했다면 어셈블리 언어의 상당한 수준을 파악하게 된셈이며 앞으로 어떤 복잡한 어셈블리 언어 프로그램도 이해 할수있는 능력이 생긴 것입니다.