제 6장 비교분기와 서브루틴

디버그(DEBUG)의 커맨드의 요약

커맨드 COMMAND	목적(Purpose)	명령 형식(Format)
Assemble	문장을 어셈블한다.	A[번지]
Compare	메모리를 비교한다.	C범위번지
Dump	메모리를 표시한다.	D[번지]이거나 D[범위]
Enter	메모리를 변경한다.	E 번지 [리스트]
Fill	메모리 블럭을 변경한다.	F범위 리스트
Go	선택적 브레이크 포인트를 갖고 실행한다.	G [=번지] [번지 [번지...]]
Hexarithmetic	16진수덧셈,뺄셈	H 값 값
Input	입력 바이트를 읽거나 표시한다.	I 포트번지
Load	화일이나 절대 디스켓 섹터를 메모리에 로드한다.	L[번지[드라이브 섹터 섹터]]
Move	메모리 블럭을 전송한다.	M 범위번지
Name	화일이나 패러미터를 정의한다.	[d:] [패스]화일이름[.확장자]
Output	출력바이트를 내보낸다.	O 포트번지 바이트
Quit	디버그 프로그램을 끝낸다.	Q
Register	레지스터나 플레그를 표시한다.	R[레지스터이름]
Search	문자를 찾는다.	S 범위리스트
Trace	집행하고 표시한다.	T [=번지][값]
Unassemble	명령을 디어셈블한다.	U[번지]이거나 U[범위]
Write	화일이나 절대 디스켓 섹터를 기록한다.	W[번지][드라이브 섹터 섹터]

6.1 반복처리와 조건분기 명령(conditional branch instruction)

조건분기 명령:반복을 할때 필요 불가결한 요소

```
CMP     AX,0  ; AX레지스터의 값을 0 과 비교하라
JE      100   ; 바로 위의 비교 결과가 같다면 100번지로 분기하라
```

CMP(compare) : 레지스터나 메모리 및 숫자의 크기를 비교합니다.
JE(Jump if Equal) 같다면 분기하라
JNE(Jump if Nor Equal) 같지 않다면 분기하라
JA(Jump if Above) ~ 이상 이라면 분기하라
JC(Jump if Carry) 캐리플래그가 1 이라면 분기하라

CMP	범용 레지스터(8/16비트)	레지스터(8/16비트)
	메모리 (8/16비트)	메모리 (8/16비트)
		숫자 (8/16비트)

예제) JMP1.ASM
1부터 100 까지 의 수를 더하는 프로그램

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
        MOV     AX,CODE     ; DS설정
        MOV     DS,AX
;
        MOV     CX,100    ;CX에 100을 지정
        MOV     AX,0      ;AX 에 0을 지정
MEXT:   ADD     AX,CX
        SUB     CX,1
        CMP     CX,0      ; CX의 내용이 0인가 비교
        JNE     NEXT      ;0이면 밑으로 아니면 NEXT 라는 라벨로 이동
        MOV     TOTAL,AX
        MOV     AH,4CH    ;끝내고 MS-DOS로 돌아간다.
        INT     21H
;
TOTAL   DW      ?
;
CODE    ENDS
        END
```

라벨(label)의 사용:
라벨이라는 것은 명령등이 있는 번지에 붙여진 이름입니다.
라벨이름의 직후에 콜론 : 을 넣는다

INC,DEC 명령:
레지스터나 메모리의 내용을 +/-1 하기위한 명령으로서 INC(increment.증가)
DEC(decrement.감소)명령이 준비되어 있읍니다.

```
        SUB     CX,1
```

대신

```
        DEC     CX
```

로 쓸수가 있읍니다.

INC 범용 레지스터(8/16비트)
메모리 (8/16비트)
DEC 범용 레지스터(8/16비트)
메모리 (8/16비트)

또한 여산 직후에 그결과가 0 인지 아닌지를 비교하여 분기하는 경우에는 비교명령을 생략 할수가 있읍니다. 플레그 레지스터의 동작에 관한 절에서 설명하겠지만 ,ADD나 SUB,INC,DEC 등의 연산 명령 다음에 는 직접 조건 분기 명령을 넣을 수가 있읍니다.
(이 경우 에 는 결과가 0 으로 비교된 것으로 간주 됩니다.)
따라서

```
NEXT:   ADD     AX,CX
        DEC     CX
        CMP     CX,0
        JNE     NEXT
```

는

```
NEXT:   ADD     AX,CX
        DEC     CX
        JNE     NEXT
```

가 됩니다.

반복을 끝내는 조건은 ,처음의 하나 ,마지막의 하나에 주의

```
 L1:    ADD     AX,CX
        INC     CX
        CMP     CX,100
        JBE
```

CX=100 으로 해버린면 AX 에는 100 이 더해지지 않읍니다.
그래서 반복하는 조건으로 CX < 100 의 사이에서가 아니라 CX<=100의 조건분기 명령에는 JBE(Jump if Below or Equal)를 사용합니다.

DUP의사명령과 블럭전송:

```
---------------------
        DB      16 DUP (?)
```

>DUP는 duplication 의 약자

여기에서 행한 정의 중에 사용되고 있는 DUP는 ()속에 것을 ()왼쪽의 갯수만큼 반복하라는 의미입니다.

예제)JMP3.ASM
MSG1부터 저장된 10바이트를 MSG2 로 전송하는 프로그램

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
;
        MOV     AX,DATA     ; DS설정
        MOV     DS,AX
        MOV     SI,OFFSET MSG1 ;SI 에 MSG1 번지값을 지정한다.
        MOV     DI,OFFSET MSG2
        MOV     CX,10
CONT:   MOV     AL,BYTE PTR [SI]  ;MSG1 부터 MSG2 로 1바이트전송
        MOV     BYTE PTR [DI],AL
        INC     SI  ;전송하는 번지를 하나씩 증가시킨다.
        INC     DI
        DEC     CX  ;전송해야할 갯수의 카운트를 하나씩 줄인다.
        JNE     CONT
        MOV     AH,4CH  ; 끝내는 시스템호출
        INT     21H
;
CODE    ENDS
;-----------------
DATA    SEGMENT
MSG1    DB      'ABCDEFGHIJKLMNOP'
MSG2    DB      16 DUP (?)  ;16바이트 만큼의 비초기화 메모리를 확보
DATA    ENDS
        END
```

데이터 세그먼트(DS;data segment)값을 아는 방법:
위프로그램의 실행결과를 확인하려면 ,MSG1 및 MSG2 에 있는 번지를 알지않으면 안됩니다. 그러나 프로그램을 실행하기전에는 DS 의 값은 PSP(program segment prefix)의 선두를 가리키고 있기 때문에 데이터를 읽을 수가 없읍니다.

DS:데이터 세그먼트 >DS,ES
레지스터 PSP
ES:엑스트라 세그먼트 >CS,SS
레지스터 코드 세그먼트
< 여기에 세그먼트 번지를
데이터 세그먼트 알고싶다

메모리에 로드되었을 때의 EXE화일의 예

실제로 데이터가 저장되어 있는 세그먼트 번지를 알필요가 있읍니다.

```
MOV     AX,DATA
MOV     DS,AX      에서 DATA 의 내용을 대치하는
MOV     AX,2244    로 역어셈블 됩니다. 2244는 각 PC 의 상황에따라 다릅니다.
```

```
-D2244:0,1F
2244:0000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50 ABCDEFGHIJKLMNOP
```

레지스터 읽는 R 커맨드에 의해 데이터 세그먼트의값을 세트한다음,다음과 같이 볼수 도있읍니다.

```
-RDS
DS 2232
:2244
-D0,1F
2244:0000 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4F 50
```

조건 분기 명령에 대한 일람표를 만들었읍니다.

분기하기위한 A,B 의 대소관계	부호없는 숫자인 경우	부호있는 숫자인 경우
A>B	JA/JNBE	JG/JNLE
A>=B	JNB/JAE	JGE/JNL
A=B	JZ/JE	JZ/JE
A<=B	JBE/JNA	JLE/JNG
A<B	JB/JNAE	JL/JNGE
A=\=B	JNZ/JNE	JNZ/JNE

6.2 무조건 분기명령

JMP 명령:

예제)JMP4.ASM

MSG1번지부터 시작하는 문자열을 MSG2번지 이후로 옮기는데 $ 자가 나타날때까지 행하는 프로그램

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
;
        MOV     AX,DATA         ;DS 설정
        MOV     DS,AX
        MOV     SI,OFFSET MSG1 ;MSG1번지를SI에 지정
        MOV     DI,OFFSET MSG2
        MOV     DX,0
CONT:   MOV     AL,BYTE PTR [SI]
        CMP     AL,"$"          ; $  이면 종료
        JE      EXITP
        MOV     BYTE PTR [DI]
        INC     SI
        INC     DI
        INC     DX              ;전송문자수의 카운트
        JMP     CONT            ;무조건 반복
EXITP:  MOV     NUMBER,DX
        MOV     AH,4CH
        INT     21H
;
CODE    ENDS
;-------------------
DATA    SEGMENT
MSG1    DB      'ABCDEFGHIJKLMNO'
MSG2    DB      16 DUP (?)
NUMBER  DW      ?
DATA    ENDS
        END
```

종료를 나타내는 문자 (여기서는 $ 자)가 나타날때 까지 전송을 반복한다.

대문자 - 소문자 변환 루틴:
영어대문자는 41H(=A)부터 5AH(=Z) 까지 연속적으로 할당되어있읍니다.

```
CODE    SEGMENT
        ASSUME  CS:CODE
MEXT:   MOV     AH,1    ; 1문자입력 시스템 호출명령사용
        INT     21H
        CMP     AL,1AH  ; CTRL+Z 이면 종료
        JE      EXITP
        CMP     AL,'A'  ; 아스키코드  A' 미만이면 L1 으로
        JB      L1
        CMP     AL,'Z'  ; 아스키코드  Z' 보다크면 L1 으로
        JA      L1
        ADD     AL,'a'-'A' ;상응하는 소문자를 찾기위해 a'-'A'=20H를 더한다.
L1:     MOV     AH,2     ; 1문자 출력 시스템 호출 명령사용
        MOV     DL,AL
        INT     21H
        JMP     NEXT
EXITP:  MOV     AH,4CH
        INT     21H
CODE    ENDS
        END
```

6.3 비교분기와 플레그 레지스터

조건분기명령은 비교명령이나 연산명령의 결과 에 의해 세트된 플레그 레지스터의 값에 따라서 분기할것인지 아닌지를 결정하고 있는 것입니다.

플레그 레지스터의 각 비트의 의미
플레그 레지스터는 이와 같이 일반적인 레지스터와 달라서 ,여러가지 상태를 기억하기 의한 레지스터입니다.
비트마다 독립된 의미를 가지고있어 독립되어 작동한다.

CF:캐리 플레그
연산한 결과,최상의 비트로부터의 높은 자리로 자리올림(캐리 CARRY) 혹은 최상의 비트로부터의 빌림(borrow)이 발생한 경우에 1로 세트되고 그 이외의 경우 0 으로 리셋된다. 그밖의 목적으로 사용될때도 있다.

PF:패리티 플레그
연산한 결과 ,1 로 된 비트의 수가 짝수개(even number)일 때 1로 세트되 고,홀수(odd number)일때 0 으로 리셋된다.

AF:보조 캐리 플레그
8(16)비트 연산에서 ,하위 4(8)비트로 부터 상위 4(8)비트로 자리올림 혹은 빌림이 발생한 경우에 1로 세트되고 그이외의 경우 0 으로 리셋된다. 10 진 보정명령에 있어서도 사용된다.

ZF:제로 플레그
연산한 결과 가 0 으로 되었을 때에 1로 세트되고,그 이외일때에 0 으로 리셋된다.

SF: 사인 플레그
연산한 결과,최상의 비트가 1 이 되었을 때 (즉 ,보수표현으로 음수가 되었을 때) 1로 세트되고,그 이외일때에 0 으로 리셋된다.

OF:오버플로우 플레그(overflow flag)
연산을 부호가 달린 숫자로 했을 때 ,오버플로우 혹은 언더 플로우(under flow)가 발생한 경우에 1로 세트되고,그이외 일때에는 0 으로 리셋된다.

이상의 6가지플레그는 연산의 결과에 따라서 세트 혹은 리셋되는 플레그로써 스테이터스 플레그(status flag, 상태 플레그)라고 부릅니다.
플레그에는 그 이외에도 3가지더 있어서 ,그들은 CPU 의 동작 상태를 제어하는 것으로서 컨트롤 플레그(control flag,제어 플레그)라고 부릅니다.

DF:디렉션 플레그(direction flag)
스트링 조작을 할때에 이 플레그가 0 이면 번지를 나타내는 레지스터 값이 자동적으로 증가하고 1 이면 레지스터값은 자동적으로 감소한다.

IF:인터럽트 플레그(interrupt . enable . flag)
이 플레그가 0일 때 INTR 단자로 부터의 외부 인터럽트요구는 무시되고 ,1일때에는 외부 인터럽트 요구를 받아들일수 있게 된다.

TF:트랩 플레그(trap flag)
이 플레그가 0 일때 CUP 는 보통대로 명령을 실행한다.
이 플레그가 1 일때 CUP 는 한 명령 시행할때 마다 자동적으로 내부 인터럽트(INT1)

를 발생하고 ,인터럽트처리 루틴으로 들어간다.(단,이 처리 루틴의 실행중에는 트랩플래그는 0 으로클리어된다.) 프로그램의 추적(trace)에 사용된다.

캐리 플레그는 프로그램에 의해서도 세트 혹은 리세트 할 수 있습니다.

플레그 레지스터 제어명령

STC(Set Carry flag)	캐리플레그를 1로 세트한다.
CLC(CLdar Carry flag)	캐리 플레그를 0으로 리셋한다.
CMC(CoMplement Carry flag)	캐리 플레그가 0이면 1로 ,1이면 0으로 세트한다.
STD(Set Direction flag)	디렉션 플레그를 1로 세트한다.(번지를 자동적으로 감소시키는 방향)
CLD(CLear Direction flag)	디렉션 플레그를 0 으로 리셋한다.(번지를 자동적으로 증가시티는 방향)
STI(Set Interrupt-enable flag)	인터럽트 플레그를 1 로 세트하고 인터럽트를 받아들일수 있는 상태로 한다.
CLI(CLear Interrupt-enable flag)	인터럽트 플레그를 0으로하고,인터럽트를 걸지 못하게 금지상태로 한다.
LAHF(Load AH from Flags)	플래그 레지스터의 하위 8비트를 AH레지스터로전송한다.사용하지 않는 비트의 값은 정해져 있지않다.
SAHF(Store AH into Flags)	AH레지스터의 내용을 플래그 레지스터의 하위 8비트에 전송한다.
PUSHF(PUSH Flags onto stack)	플레그 레지스터의 내용을 스택(stag)상에 푸쉬(PUSH)한다.
POPF(POP Flags off stack)	스텍위의 워드 데이터를 플레그 레지스터에 POP 한다.

비교연산 명령과 조건분기명령의 관계:
비교연산 명령은 뺄셈명령과 같은 것입니다.다른점은 비교한 결과 비교를 행한 레지스터나 메모리의 내용은 전혀 변하지 않고 단지 플레그 레지스터만을 변화시킨다.
CMP AX,BX
라고 했을때,AX 레지스터로 부터 BX 레지스터의 내용을 (가상적으로) 뺀다음,그 결과에 따라서 플레그 레지스터 만을 변화 시키고 AX,BX 의 내용은 보존 됩니다.
SUB AX,BX
라고 하면 플레그 레지스터는 완전히 똑같게 세트되지만,AX 레지스터의 값도 변화되어 AX 에서 BX 를 뺀 결과가 AX 의 값이 되어 버립니다.
플레그레지스터의 값은 산술 논리연산을 행할 때에 세트 리셋되고, 전송명령이나 분기명령을 실행해서는 변화하지 않습니다.

조건 분기 명령의 정리

A,B 의 대소 관계	부호없는 숫자의 경우	조건을 만족하는 플레그의 상태 CF		부호있는 숫자의 경 우	조건을 만족하는 플레그의 상태 SF OF		ZF
A>B	JA/JNBE	0	0	JG/JNLE	0	0	
A>=B	JNB/JAE	0	*	JGE/JNL	0	*	
A=B	JZ/JE	*	1	JZ/JE	*	1	
A<=B	JBE/JNA	1	1	JLE/JNG	1	1	
A<B	JB/JNAE	1	*	JL/JNGE	1	*	
A=\=B	JNZ/JNE	*	0	JNZ/JNE	*	0	

A,B 의 대소관계는 ,CMP A,B 즉(A-B)의 결과로 표시된다.

는 배타적 논이합(exclusive OR)을 구하는 연산이다.
* 표시란의 값은 0 이라도 1 이라도 조건판단에 영향을 주지 않는다.

JA/JNBE 와 같이 / 로 구분된 2개의 표현은 ,완전히 같은 관계를 다른 말로 표현한 것으로서 ,어느쪽을 사용해도 상관이 없다.
단,디버그에 의한 디스어셈블 결과는 왼쪽의 표현 으로 된다.
조건 분기명령에서는 쇼트 점프(short jump) 즉,명령이 있는 다음 번지로 부터 +127 ~ -128바이트 이내로 만 분기할수있다.

플레그 1일때의 분기	0일때의 분기	플레그가 1일때의 의미
SF JS	JNS	보수표현으로 음수이다.
ZF JZ/JE	JNZ/JNE	결과가 0 이다.(위표의 A=B와 같다)
PF JP/JPE	JNP/JAE	1이된 비트의 수가 짝수이다.
CF JB/JNAE	JNB/JAE	캐리플레그가 1이다(연산한결과 자리올림 혹은 빌림이 발생한 경우 등)
OF JO	JNO	보수표현을 사용한 연산에서 결과가 범위를 초과했다.

레지스터		명령의 의미
CX	------ JCXZ	CX레지스터가 0일때만 분기한다.

조건분기명령 약자를 읽는법

A:Above	위(부호없음)	O:Overflow	오버플로우
B:Below	아래(부호없음)	Odd	짝수
E:Equal	0 혹은 같음	P:Parity	패리티
G:Greater	위(부호있음)	S:Sign	부호가 음수
L:Less	아래(보호있음)	Z:Zero 0	혹은 같음
N:Not	~ 가아니다		

JNGE 는
Jump if Not Greater or Equal
이므로(A>=B),즉 A<B라는 의미가 됩니다.

플레그레지스터 표시를 보는법:
DEBUG에서는 ,R 커맨드로 레지스터의 내용을 표시했을 때 그리고 추적(trace) 고우(GO)인 T,G 커맨드 로 실행을 추적했을 때에,각종 레지스터의 내용과 함께 플레그레지스터의 내용도 표시 됩니다.플레그 레지스터의 표시는 숫자나 1,0 에 의한 것이 아니라 각 플레그 의 명칭을 사용 하여 표시 됩니다.
각 플레그의 0,1 의 상태와 표시를 대응시킨 것은 다음과 같습니다.

플레그 이름	OF	DF	IF	SF	ZF	AF	PF	CF
0	NV	UP	DI	PL	NZ	NA	PO	NC
1	OV	DN	EI	NG	ZR	AC	PE	CY

플레그 이름	0	1
Overflow flag	Not oVerflow	OVerflow
Direction flag	UP	DowN
Interrupt flag	Disable Interrupt	Enable Interrrupt
Sign flag	PLus	NeGative
Zero flag	Not Auxiliary carry	ZeRo
Auxiliary Carry flag	Parity Odd	Auxiliary Carry
Parity flag	Parity Odd	Parity Even
Carry flag	Not Carry	Carry

```
MOV     BX,8888H
CMP     BX,0
```
0과 비교한결과 8888H - 0H의 결과에 따라서 플래그 레지스터가 세트됩니다.
뺄셈의 결과 는 8888H 이므로 최상위 비트가 1 이 되고 ,사인 플래그는 NG(음수)가 됩니다.그러나 빌림은 발생하지 않았으므로 ,캐리 플래그는 NC (0)가 되어 있읍니다.
오버플로우 플래그가 NV(0)입니다.
따라서 부호없는 비교 JB 는 조건을 만족하지 않으므로,다음 명령으로 진행합니다.
여기에서 부호있는 비교 JL은 조건을 만족하고 있으므로 ,지정한 번지로 분기 합니다.

주의)
1을 더하는 명령을 ADD명령으로 한경우와 ,INC 명령으로 한 경우
덧셈결과는 양쪽다 같지만 플래그 레지스터의 변화는 달라 집니다.
보통의 산술연산 명령에서는 자리 올림 혹은 빌림이 발생한 경우는 캐리 플래그가 세트 되지만 ,INC 명령과 DEC 명령만은 특수해서,이 2 종류의 명령만은 자리올림이나 빌림이 발생해도 캐리 플래그에 영향을 주지 않습니다.
이때문에 자리올림()을 사용하여 조건판단을 하는 명령앞에 INC 명령,DEC 명령을 놓을 때 는 주의 가 필요 합니다. 캐리 플래그는 ,이들명령보다 앞의 상태를 계속 가지고 있읍니다.
INC ,DEC 명령이 이와 같은 성질을 갖고있다는 것은 결코 불편한것이 아니라 ,오히려 실제 프로그램 상에서 느 도움이 되는 경우가 많읍니다.
캐리 플래그를 제외한 다른 플래그 레지스터는 INC,DEC 명령의 결과에 따라서 세트 혹은 리셋됩니다.

6.4 서브루틴의 사용법

서브루틴(subroutine)이란
프로그램 중에서 여러번 되풀이 사용되는 부분을 메인 루틴으로 부터 독립하여 작성 프로그램을 효율적으로 하는 효과가있다.
프로그램을 처리내용에 따라 불럭으로 나눔으로써 전체를 보기 쉽도록 해주고 ,보수를 쉽게 만들어 주는 이점이 있읍니다.
어셈블러의 서브루틴은 BASIC 과 비슷한 사용방법이다.
서브루틴의 끝을 나타내는 RET 명령으로 끝납니다.
RET (return command)는 서브루딘으로 부터 메인 루틴으로 돌아오는 명령입니다.
서브루틴을 부를 때는 CALL 다음에 서브루틴의 시작에 붙여진 라벨을 사용합니다.
예)CALL1.ASM
0부터 9까지의 숫자를 출력하는 프로그램이다. 단 한 자 한자 출력하는 서브루틴을 만들어 호출한다.

```
CODE    SEGMENT
        ASSUME  CS:CODE
;
```

```
        MOV     CX,0
NEXT:   CALL    SUB1    ;CALL명령으로 서브루틴을 호출
        INC     CX
        CMP     CX,10
        JB      NEXT
        MOV     AH,4CH
        INT     21H
;
SUB1:   MOV     DX,CX
        ADD     DL,'0'
        MOV     AH,2
        INT     21H
        RET         ;리턴명령
CODE    ENDS
        END
```

서브루틴 SUB1속에서는 , CX 레지스터의 값을 변경시키지 않도록 CX 레지스터의 값을 DX 레지스터로 전송한다

16진수값의 화면상에 출력법:

기본구상 _ AL레지스터의 내용을 표시하려고 한다면, AL=3FH일경우 "3" 와 "F" 와의 2가지 문자를 출력하지 않으면 안된다.

AL레지스터의 내용을 상위 4비트와 하위 4비트로 나누어서 ,2번에 걸쳐 출력해야 한다.

```
   AL          DL

  3  F      > 0  F       >아스키 코드로 변환   > 출력

              DL

           > 0  3       >아스키 코드로 변환   >  출력
```

레지스터의 비트 단위로 데이터를 취급하려면,보통 논리 연산명령이나,쉬프트(이동) 로테이트(rotate, 회전) 명령을 사용한다.

여기서는 학습을 위해 곱셈과 나눗셈을 사용한다.

```
                    AL          3F
                                \/ * 10H
                    AX          03F0
                                /
          상위자리 보존 <     03
                    AH        00
                                \ \/
                    AX          00F0
                                 \/  %10H
                    AX          000F
                                  \/
                               <    0F
```

예제)PUTAL.ASM
AL레지스터 내의 값을 ASCII코드 16진수값으로 화면에 표시하는 프로그램
(서브루틴을 사용한다.)AL 내의 값은 3FH값을 예로 들었다.

```
CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE
;
        MOV     AX,CODE  ; DS의 초기치 설정
        MOV     DS,AX
;
        MOV     AL,3FH   ;처음AL에 3FH가 들어 있다고 하자
        CALL    PUTAL    ;서브루틴을 호출한다.
;
        MOV     AH,4CH
        INT     21H
;-----------------------
PUTAL:  MOV     BL,10H    ;AL을 10H 하면 AH,AL                    상위자리
        MUL     BL        ;              03 F0 가 된다.           하위자리로
        MOV     LEVEL2,AH ;                                      분할하는
        MOV     AH,0      ;AX를 10H 분의 1 로 나누면 AH AL       서브루틴
        MOV     BL,10H                          00 0F 가 된다
        DIV     BL
        MOV     LEVEL1,AL  ; 하위자리 보존
        MOV     DL,LEVEL2  ;    상위자리 표시
        CALL    PUTHEX
        MOV     DL,LEVEL1  ;     하위자리 표시
        CALL    PUTHEX
        RET
;---------------------------
PUTHEX: CMP     DL,0AH
        JAE     HEX2
        ADD     DL,'0'    ; 0 의 ASCII 코드 값이 30H 이다.      수치에 대한
        JMP     HEX3          (0 에서 9 인경우)                  아스키 코드를
HEX2:   ADD     DL,'A'-0AH ;   A 에서 F 인경우                   산출,표시하는
HEX3:   MOV     AH,2           'A  의 ASCII코드값은 41이다       서브루틴
        INT     21H            'B'는 42가 표시도기 위해서는
        RET                    41+1이 되어야 하는데 이것은
                               A'+(BH-AH)의 계산으로 가능하다.
LEVEL2  DB      ?
LEVEL1  DB      ?
;
CODE    ENDS
        END
```

서브루틴 PUTAL 속에서 다시 서브루틴 PUTHEX 를 호출할수도 있읍니다.
여기에서 출력시스템 INT 21은 DL 레지스터에 ASCII코드값을 넣을 때 그에 해당하는 ASCII 문자가 화면에 출력된다.
0 에서 9 까지의 숫자에 대응하는 아스키 코드와 A 에서 F 까지 대응 하는 아스키코드가 연속하여 있지안으므로 ,숫자가 0AH 미만인지 아닌지 에 따라 변환을 위해서 더하는 숫자가 달라 집니다.

문자	0 ~ 9	A ~ F
아스키 코드	30 ~ 39	41 ~ 46

숫자를 표시할경우에는 윗자리를 먼저, 아랫자리를 나중에 표시하지 않으면 안 됩니다. 구한 순서와 표시한 순서가 일치하지 않을 때도 있으므로 ,그 와 같은 경우에는 각 자리의 숫자를 보존해 두는 것이 필요합니다.

PROC에 의하 서브루틴의 구조화:
서브루틴을 보다 구조적으로 표현하기위해서 프로시듀어(procedure)의사 명령에 의해 서브루틴의 앞뒤를 명시 하도록 합니다.

```
SUB1    PROC
    서브루틴 본체
SUB1    ENDP
```

16진수값의 입력법
예제)GETAL.ASM
키보드로 부터 ASCII코드를 입력하여 이코드에 해당하는 ASCII문자 출력

```
CODE    SEGMENT
        ASSUME  CS:CODE
CR      EQU     0DH
LF      EQU     0AH
;
NEXT:   CALL    GETAL  ; 키보드로 부터 두개의 숫자 입력
        JC      EXITP  ;에러이면 끝냄
        MOV     DL,AL
        CALL    PUTASC ;아스키 코드의 출력
        JMP     NEXT
EXITP:  MOV     AH,4CH  ;종료하여 MS-DOS로돌아간다.
        INT     21H
;-----------------------
GETAL   PROC
        MOV     AH,1   ; 우선한자를 입력한다.
        INT     21H
        CALL    CVTAL   ; 아스키 코드    > 숫자
        JC      GETEND  ;에러이면 되돌아 간다.
        MOV     DH,AL   ; 보존
        MOV     AH,1    ;두번째글자입력
        INT     21H
        CALL    CVTAL      아스키 코드    > 숫자
        JC      GETEND  ;에러이면 되돌아간다.
        MOV     DL,AL
        MOV     BH,10H  ;상위자리를 10배
        MOV     AL,DH
        MUL     BH
        ADD     AL,DL   ;하위자리와 더한다
GETEND: RET
GETAL   ENDP
;--------------------------
CVTAL   PROC
        CMP      AL,'0'
        JB      ERR1       ;ASCII 코드값으로부터
        CMP     AL,'9'      30H보다 적을경우 ERR1
        JA      SKIP1
        SUB     AL,'0'
```

```
        JMP     SKIP2
SKIP1:  CMP     AL,'A'          'A'~'F'이면
        JB      ERR1            아스키 코드의
        CMP     AL,'F'          'A'를 빼고
        JA      ERR1            0AH를 더한다.
        SUB     AL,'A'-0AH
SKIP2:  CLC                     ;에러가없으면
        JMP     CVTEND          캐리를 클리어
ERR1:   STC                     ;에러이면 캐리를 세트
CVTEND: RET
CVTAL   ENDP
;---------------------
PUTASC  PROC
        MOV     DH,DL           ; DL 에있는 ASCII코드 16진 값을잠깐 대피시킨다.
        MOV     DL,' '          ; DL에 있는 아스키코드 16진값을 잠깐대피시킨다.
        MOV     AH,2        공백출력
        INT     21H
        MOV     DL,DH           ; 대피시켰던것을 되 찾는다
        MOV     AH,2        아스키코드 출력
        INT     21H
        MOV     DL,CR           ; carriage return 을 출력
        MOV     AH,2         복귀 개행
        INT     21H
        MOV     DL,LF           ; line feed
        MOV     AH,2
        INT     21H
        RET
PUTASC  ENDP
CODE    ENDS
        END
```

CVTAL은 ,GETAL로부터 호출된 서브루틴에서 아스키 코드를숫자로 변환하는것입니다.
만일 숫자이외에 입력이 있을 때에는 캐리플래그가 세트되어 돌아옵니다.
PUTASC 는 1문자를 띄고 지정된 아스키 코드에 상당하는 문자를 출력하고 ,줄을 바꾸는것입니다.
GETAL에서는 캐리플레그가 세트되어 있는지 어떤지를 체크하여,세트되어있으면 에러로 간주 메인루틴으로 돌아옵니다.
메인 루틴에서도, 역시 캐리플래그가 세트되어 있다면 에러로 간주하고 처리를 끝냅니다. 캐리 플래그에 의해서 분기하는 명령이 JC명령입니다.