Arithmetic/Logic Instructions

- Basic Mathematical Operations
  - Signed/Unsigned Integer Only
  - Default is 2's Complement
  - Computes Result *AND* Modifies Status Flags
- Logic Instructions
  - Bit Level
  - Word Level
  - Computes Results *AND* Modifies Flags

Arithmetic Instruction Summary

```
add     ax,   bx            ;ax<-ax+bx and set flags
adc     ax,   bx            ;ax<-ax+bx+CF(lsb) and set flags
inc     ax                  ;ax<-ax+1 and set flags
aaa                         ;ASCII Adjust after Addition
daa                         ;Decimal (BCD) Adjust after Addition
sub     ax,   bx            ;ax<-ax-bx and set flags
sbb     ax,   bx            ;ax<-(ax-CF)-bx and set flags
dec     ax                  ;ax<-ax-1
neg     ax                  ;ax<-(-1)*(ax) - 2's Complement
cmp     ax,   bx            ;ZF is set according to ax-bx
das                         ;Decimal (BCD) Adjust after Subtraction
aas                         ;ASCII Adjust after Subtraction
mul     cx                  ;dx:ax<- ax * cx (unsigned)
imul    cx                  ;dx:ax<- ax * cx (2's complement)
aam                         ;ASCII Adjust after Multiplication
div     cl                  ;al<-ax/cl Quot. AND ah<-ax/cl Rem.
idiv    cx                  ;ax<-(dx:ax)/ax Quot. AND dx <- Rem.
aad                         ;ASCII Adjust after Division
```

Addition Instruction Types

```
add     ax,   bx              ;ax<-ax+bx and set flags
adc     ax,   bx              ;ax<-ax+bx+CF(lsb) and set flags
inc     ax                    ;ax<-ax+1 and set flags
aaa                           ;ASCII Adjust after Addition
daa                           ;Decimal (BCD) Adjust after Addition

add     al,   bl              ;al<-al+bl and set flags
add     bx,   35afh           ;bx<-bx+35afh
add     [bx], al              ;ds:bx<-ds:bx+al
add     cl,   [bp]            ;cl<-cl+ss:bp
add     al,   [ebx]           ;al<-al+ds:ebx
add     bx,   TEMP[di]        ;bx<-bx+ds:(TEMP+di)
add     bx,   [eax+2*ecx]     ;bx<-bx+ds:(eax+(2*ecx))
```

Scaled Index Addressing: 386+
ecx may contain 1, 2 , 4 only

Increment Examples

```
inc     bl                    ;bl<-bl+1 and set flags
inc     BYTE POINTER [bx]     ;Byte at ds:bx<-ds:bx+1
```

New MASM Directive: **BYTE POINTER**

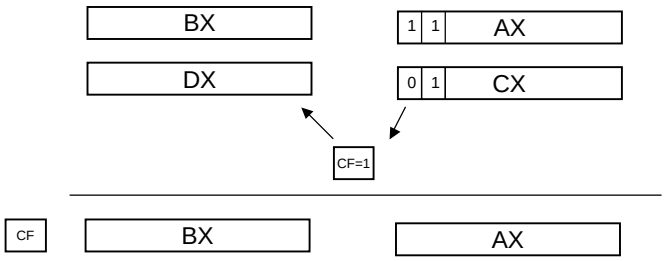
00ffh → 0000h

```
inc     [bx]                  ;Word at ds:bx<-ds:bx+1
```

00ffh → 0100h

```
inc     DATA1                 ;ds:DATA1<-ds:DATA1+1
```

Add with Carry




```
add     ax,   cx        ;ax<-ax+cx and flags set
adc     bx,   dx        ;bx<-bx+dx+CF(lsb) and flags set
```

33-bit Sum Present in `CF:bx:ax`

Decimal Adjust after Addition

- For BCD Arithmetic
- "Corrects" Result

```
  0110      6
+ 0111    + 7
  1101      13→should be 0001 0011
```

(1101 is illegal BCD)

- 2 Digits/Word Intel Refers to as "Packed Decimal"
- `daa` Uses Implicit Operand, `al` Register
- Follows `add`, `adc` to "Adjust"

Decimal Adjust after Addition Example

```
mov     dx,     1234h   ;dx<--1234 BCD
mov     bx,     3099h   ;bx<--3099 BCD
mov     al,     bl      ;al<--99 BCD
add     al,     dl      ;al<--cdh illegal BCD, need 34+99=133
daa                     ;al<--33h (33 BCD) and CF=1
mov     cl,     al      ;cl<--33 BCD
mov     al,     bh      ;al<--30 BCD
adc     al,     dh      ;al<--30h+12h+1=43h
daa                     ;al<--43h (43 BCD) not illegal BCD this time
mov     ch,     al      ;cx=4333h BCD for 1234+3099
```

ASCII Adjust after Addition

- **For Addition Using ASCII Encoded Numbers**

  30h through 39h Represent ‘0’ through ‘9’

- **ax is Default Source and Destination for aaa**

```
  31   ‘1’
 +39   ‘9’
  6a   ‘10’→should be 3130h
```

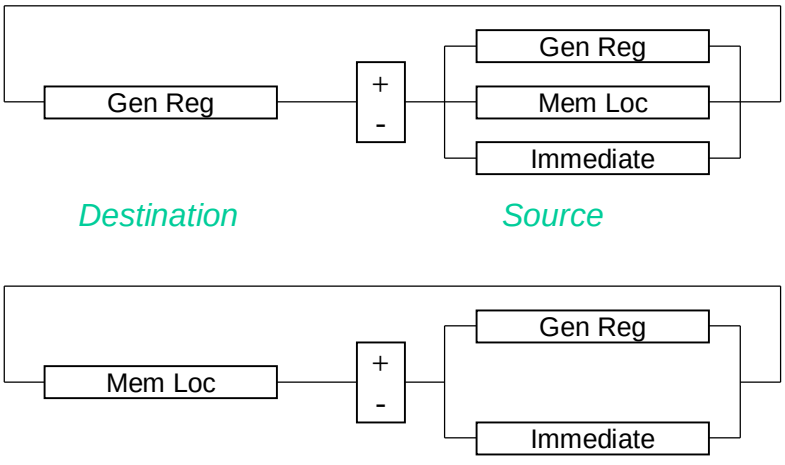
(6ah is incorrect ASCII result ‘j’)

```
mov     ax,     31h     ;ax<--0031h=‘1’
add     al,     39h     ;ax<--31h+39h=006ah=‘<nul>j’
aaa                     ;ax<--0100h (this is BCD of result)
add     ax,     3030h   ;Convert from BCD to ASCII
                        ;ax<--0100h+3030h=3130h=‘10’
```

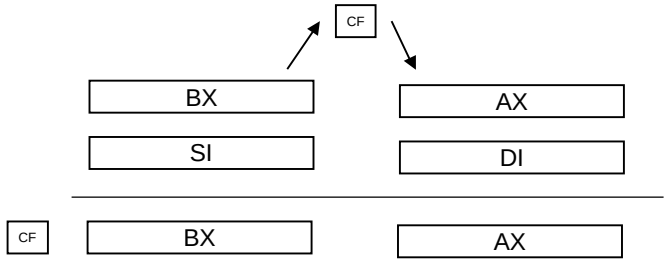
Subtraction Instruction Types

```
sub     ax,   bx              ;ax<-ax-bx and set flags
sbb     ax,   bx              ;ax<-(ax-CF)-bx and set flags
dec     ax                    ;ax<-ax-1
neg     ax                    ;ax<-(-1)*(ax) - 2's Complement
cmp     ax,   bx              ;ZF is set according to ax-bx
das                           ;Decimal (BCD) Adjust after Subtraction
aas                           ;ASCII Adjust after Subtraction
```

Allowable Operands for **add**, **sub**



Subtract with Borrow, **sbb**




```
sub     ax,   di       ;ax<--ax-di and CF gets borrow bit
sbb     bx,   si       ;bx<-(bx-CF(lsb))-si and flags set
```

32-bit Difference Present in **bx:ax**
CF Indicates If Difference is Negative

Multiplication

- 8086/8088 One of First to Include **mul**/**div** Instruction
- Allowable Operands: Bytes, Words, DoubleWords
- Allowable Results: Words, DoubleWords, QuadWords
- **OF**, **CF** Give Useful Information
- **AF**, **PF**, **ZF**, **SF** Change but Contents Unpredictable
- Multiplicand Always in **al**, **ax**, **eax**
- **mul** - Unsigned Mnemonic
- **imul** - Signed Mnemonic

Multiply Instructions

- Product can be Twice the Size
  - $2 \times 3 = 6$ (same size)
  - $2 \times 8 = 16$ (double size, EXT)
- **OF=CF=0** means product is same size as result (faster)
- **OF=CF=1** means EXT product size (slower)
- **AF**, **PF**, **ZF**, **SF** Contents Unpredictable

```
mul     bl      ;ax<--al*bl, Unsigned
mul     bx      ;dx:ax<--bx*ax, Unsigned
mul     ebx     ;edx:eax<--ebx*eax, Unsigned
imul    bl      ;ax<--al*bl, Signed
imul    bx      ;dx:ax<--bx*ax, Signed
imul    ebx     ;edx:eax<--ebx*eax, Signed
```

Special Immediate Multiply Instruction

- 286+
- Uses **imul** Mnemonic but with 3 Operands
  - first: 16-bit dest. register
  - second: reg/mem location
  - third: 8/16-bit immediate value
- Always Performs Signed Multiplication
- Product is Limited to 16-bits

```
imul   cx, dx, 12h          ;cx<--dx*12h
imul   bx, NUMBER, 1000h    ;bx<--ds:NUMBER*12h
```

Division

- 8, 16, 32 bit Operands (32 bit is 386+)
- No Immediate Addressing Mode
- No Flag Bits Change Predictably
- Can Cause Two Types of Error:
  1) Divide by 0 (Mathematically Undefined)
  2) Divide Overflow (Wordlength Problem)
- Operands: Divisor is Programmer Specified
- Dividend is Implied
- Quotient, Remainder Implied

Size	Dividend	Quotient	Remainder
8 bits	**ax**	**al**	**ah**
16 bits	**dx:ax**	**ax**	**dx**
32 bits	**edx:eax**	**eax**	**edx**

Division Instruction Examples

- **idiv** Signed and **div** Unsigned

  dividend / divisor = quotient, rmdr

```
div    cx     ;dx:ax is divided by value in cx
              ;unsigned quotient is placed in ax
              ;positive remainder is placed in dx

idiv   ebx    ;edx:eax is divided by value in ebx
              ;signed quotient is placed in eax
              ;remainder (ALWAYS same sign as
              ;dividend) is placed in edx
```