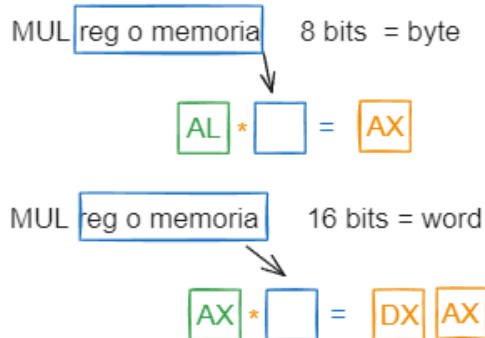


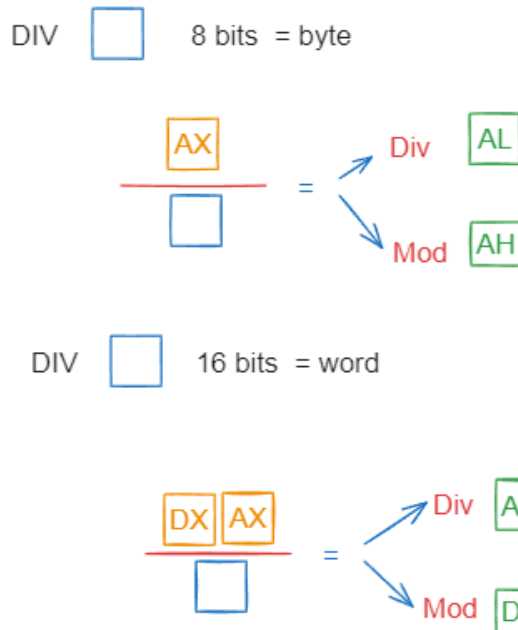
## MUL reg o memoria



when operand is a **byte**:  
 $AX = AL * \text{operand}$ .

when operand is a **word**:  
 $(DX\ AX) = AX * \text{operand}$ .

## DIV reg o memoria



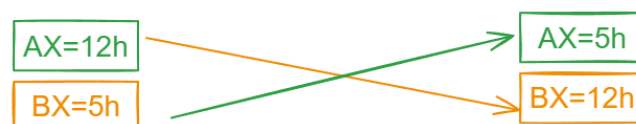
when operand is a **byte**:  
 $AL = AX / \text{operand}$   
 $AH = \text{remainder (modulus)}$

when operand is a **word**:  
 $AX = (DX\ AX) / \text{operand}$   
 $DX = \text{remainder (modulus)}$

## XCHG

XCHG	REG, memory memory, REG REG, REG
------	--

XCHG AX,BX



Leer un numero con mas de un digito

```

01 data segment
02     val10 dw 10
03     n dw 0
04 ends
05
06 stack segment
07     dw 128 dup(0)
08 ends
09 code segment
10 start:
11     mov ax, data
12     mov ds, ax
13     mov es, ax
14
15     mov n, 0
16     LeendoN:
17         mov ah, 1
18         int 21h
19         cmp al, 13
20         je finLeer
21         sub al, 48
22         mov ah, 0
23         xchg ax, n
24         mul val10
25         add ax, n
26         mov n, ax
27         jmp LeendoN
28
29     finLeer:
30
31     mov ax, 4c00h ; exit to operating system.
32     int 21h
33 ends
34
35 end start

```

123

$n=0$

$n=0*10+1$

$n=1*10+2$

$n=12*10+3=123$

ax=ahal=0ah INPUT

$n*10+input$

Reutilizar el código anterior para leer dos números o más números

```

data segment
    val10 dw 10
    n dw 0
    n1 dw 0
    n2 dw 0
    enter db 10,13,"$"
ends
stack segment
    dw 128 dup(0)
ends
macro read parms
    call leerNumero
    mov ax, n
    mov parms, ax
endm
code segment
start:
    mov ax, data
    mov ds, ax
    mov es, ax
    read n1
    lea dx, enter
    mov ah, 9
    int 21h
    read n2
    mov ax, 4c00h ; exit to operating system.
    int 21h
ends

```

```

leerNumero:
  mov n ,0
LeendoN:
  mov ah,1
  int 21h
  cmp al, 13
  je finLeer
  sub al ,48
  mov ah,0
  xchg ax, n
  mul val10
  add ax, n
  mov n,ax
  jmp LeendoN
finLeer:
ret
end start

```

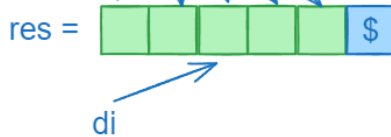
Mostrar un numero con mas de un digito en la pantalla

AX=FFFF=65535

```

res db 5 dup(' ','>','$')
val10 dw 10

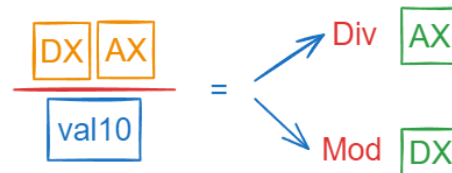
```



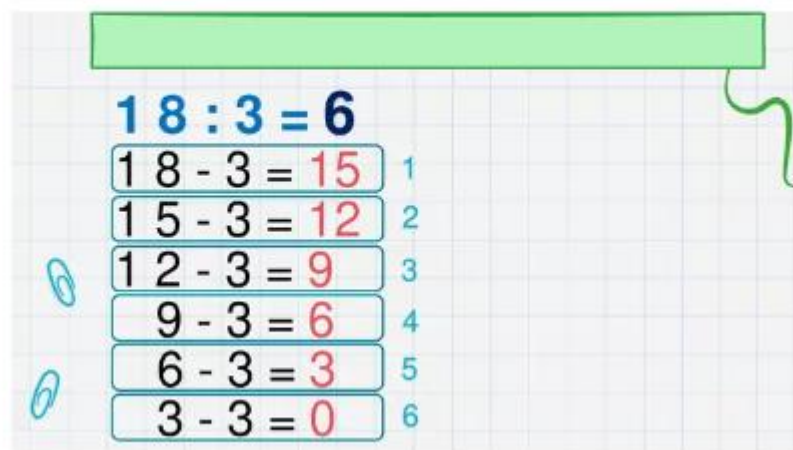
```

59  mostrar:
60      ; <ax> ->
61      mov di, 4
62      descomp:
63          mov dx, 0
64          div val10
65          add dl, 30h
66          mov res[di], dl
67          dec di
68          cmp ax, 0
69          jg descomp
70          inc di
71          mostrarCadena res
72      ret

```



Realizar division con restas sucesivas

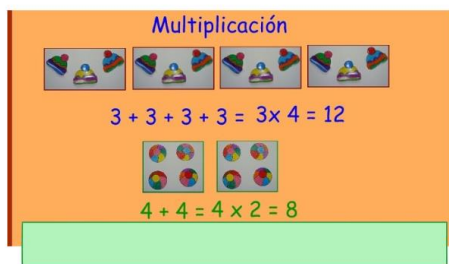


```

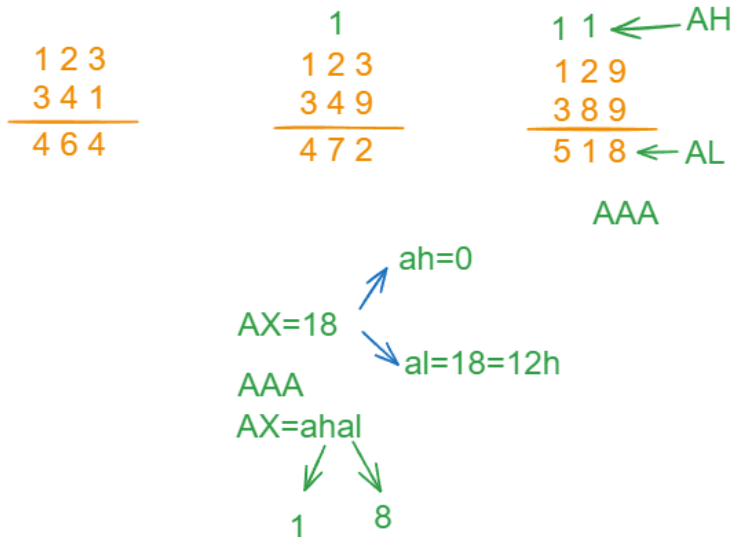
01 data segment
02     a dw 2302
03     b dw 34
04     valDiv dw 0
05     valMod dw 0
06     res db 5 dup(" "), "$"
07     val10 dw 10
08     enter db 10,13, "$"
09 ends
10
11 stack segment
12     dw 128 dup(0)
13 ends
14
15 macro mostrarCadena txt
16     lea dx, txt
17     mov ah, 9
18     int 21h
19 endm
20
21 code segment
22 start:
23     mov ax, data
24     mov ds, ax
25     mov es, ax
26     ; a/b
27     ; a-b = ax-b
28     mov valDiv, 0
29     mov ax, a
30     ciclo:
31         cmp ax, b
32         jl salir
33         sub ax, b
34         inc valDiv
35         jmp ciclo
36     salir:
37     mov valMod, ax
38     mov ax, valDiv
39     call mostrar
40     mostrarCadena enter
41     mov ax, valMod
42     call mostrar
43     mostrarCadena enter
44     mov ax, 4c00h ; exit to operating system.
45     int 21h
46 ends
47
48 mostrar:
49     ; <ax> ->
50     mov di, 4
51     descomp:
52         mov dx, 0
53         div val10
54         add dl, 30h
55         mov res[di], dl
56         dec di
57         cmp ax, 0
58         jg descomp
59         inc di
60     mostrarCadena res
61     ret
62 end start ; set entry point and stop the assembler.
63

```

Multiplicación con sumas sucesivas



## Suma de dígito por dígito



```

01 ; multi segment executable file template.
02
03 data segment
04
05     n1 db 1,2,9,"$"
06     n2 db 3,8,9,"$"
07     res db 0,0,0,0,"$"
08 ends
09
10 stack segment
11     dw 128 dup(0)
12 ends
13
14 code segment
15 start:
16 ; set segment registers:
17     mov ax, data
18     mov ds, ax
19     mov es, ax
20
21
22
23     mov al, n1[2]
24     add al, n2[2]
25     mov ah, 0
26
27     aaa
28
29     mov res[3], al
30
31     mov al, ah
32     add al, n1[1]
33     add al, n2[1]
34     mov ah, 0
35
36     aaa
37     mov res[2], al
38
39     mov al, ah
40     add al, n1[0]
41     add al, n2[0]
42     mov ah, 0
43     aaa
44     mov res[1], al
45     mov res[0], ah
46
47
48     mov ax, 4c00h ; exit to operating system.
49     int 21h
50 ends
51
52 end start ; set entry point and stop the assembler.
53

```

## Multiplicacion de digito por digito

```

  1 2 3
  3 4 1
  ---
 4 6 4

```

```

  2 8 ← AH
  1 2 9
    9
  ---
 1 1 6 1 ← AL

```

AAM

```

AX=81
AAM
AX=ahal
  ↓   ↓
 1    8

```

AAM

No operands

ASCII Adjust after Multiplication.  
Corrects the result of multiplication of two BCD values.

Algorithm:

- AH = AL / 10
- AL = remainder

Example:

```

MOV AL, 15 ; AL = 0Fh
AAM        ; AH = 01, AL = 05
RET

```

C	Z	S	O	P	A
?	?	?	?	?	?

```

03 data segment
04     n1 db 1,2,9,"$"
05     n2 db 9,"$"
06     res db 0,0,0,0,"$"
07     auxi db 0
08 ends
09 stack segment
10     dw 128 dup(0)
11 ends
12 code segment
13 start:
14 ; set segment registers:
15     mov ax, data
16     mov ds, ax
17     mov es, ax
18
19     mov ah, 0
20     mov al, n1[2]
21     mul n2[0]
22     aam
23
24     ; ah mod
25     ; al div
26     mov res[3], al
27
28     mov auxi, ah
29     mov al, n1[1]
30     mul n2[0]
31
32     mov dh, 0
33     mov bl, auxi
34     add ax, bx
35
36     ; ax = 36=ahal
37     aam
38
39     mov res[2], al
40
41     mov auxi, ah
42     mov al, n1[0]
43     mul n2[0]
44
45     mov dh, 0
46     mov bl, auxi
47     add ax, bx
48
49     ; ax = 36=ahal
50     aam
51
52     mov res[1], al
53     mov res[0], ah
54
55
56     mov ax, 4c00h ; exit to operating system.
57     int 21h
58 ends
59
60 end start ; set entry point and stop the assembler.
61

```

## Div de dígito por dígito

```

  2 4 4 | 4
- 2 4   |
  ---   |
  0 0 4 |
    - 4 |
    --- |
      (0)

```

AAD

No operands

ASCII Adjust before Division.  
Prepares two BCD values for division.

Algorithm:

- $AL = (AH * 10) + AL$
- $AH = 0$

Example:

```

MOV AX, 0105h ; AH = 01, AL = 05
AAD           ; AH = 00, AL = 0Fh (15)
RET

```

C	Z	S	O	P	A
?	r	r	?	r	?

```

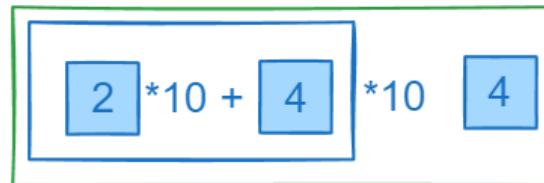
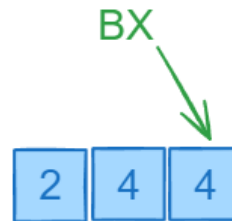
01 data segment
02     a db 2,4,4
03     b db 4
04     n db 0
05     resDiv db 0
06     resMod db 0
07     enter db 10,13,"$"
08 ends
09 stack segment
10     dw 128 dup(0)
11 ends
12 macro Mostrar txt
13     lea dx, txt
14     mov ah, 9
15     int 21h
16 endm

```

```

17 code segment
18 start:
19     mov ax, data
20     mov ds, ax
21     mov es, ax
22     mov bx, 0
23     leer:
24         mov ah, 1
25         int 21h
26         cmp al, 13
27         je finLeer
28         sub al, 30h
29         mov a[bx], al
30         inc bx
31         inc n
32         jmp leer
33     finLeer:
34
35     Mostrar enter
36
37     mov ah, 1
38     int 21h
39     sub al, 30h
40     mov b, al
41
42     mov bx, 0
43     cmp n, 1
44     je Dividir
45     mov ah, a[bx]
46     inc bx
47     mov al, a[bx]
48     aad
49     ciclo:
50         inc bx
51         cmp bx, n
52         jge FinCiclo
53         mov ah, al
54         mov al, a[bx]
55         aad
56         jmp ciclo
57
58     Dividir:
59         mov al, a[0]
60         mov ah, 0
61
62     FinCiclo:
63         div b
64         mov resDiv, al
65         mov resMod, ah
66         mov ax, 4c00h
67         int 21h
68     ends
69 end start

```



al=244

AAD

No operands

ASCII Adjust before Division.  
Prepares two BCD values for division.

Algorithm:

- $AL = (AH * 10) + AL$
- $AH = 0$

Example:

MOV AX, 0105h ; AH = 01, AL = 05  
AAD ; AH = 00, AL = 0Fh (15)  
RET

C	Z	S	O	P	A
?	?	?	?	?	?