

Ejercicio 1 de macros internas, conociendo las macros

inicio macro ;declaramos la macro, le damos el nombre de inicio

mov ax,data ;Cargamos el segmento de datos.

mov ds,ax

mov dx,ax

endm

.model small

.stack 64

.data

msj db "Este es mi primer macro",10,13,"\$"

.code

inicio ;Llamamos al macro, lo unico que hace es cargar msj del segmento de datos.

mov ah,09h

lea dx,msj ;puede ser mov dx,offset msj

int 21h ;interrupcion

mov ax,4c00h ;Sale del programa

int 21h ;interrupcion

end

5th emulator screen (80x25 chars)

```
Este es mi primer macro
```

Ejercicio 2 Macros Internas

restas macro p, s

mov al, p

sub al, s

add al, 30h

mov resta, al

mov ah, 09

lea dx, msj3

int 21h

mov ah, 02

mov dl, resta

int 21h

mov ah, 4ch

int 21h

endm

.model small

.stack 64

.data

n1 db 0

n2 db 0

resta db 0

msj db "Dame el primer valor: \$"

msj2 db 10,13, "Dame el segundo valor \$"

msj3 db 10,13, "resta = \$"

.code

mov ax, @data

mov ds, ax

mov ah, 09

lea dx, msj

int 21h

mov ah, 01

int 21h

sub al, 30h

mov n1, al

mov ah, 09

lea dx, msj2

int 21h

mov ah, 01

int 21h

sub al, 30h

mov n2, al

restas n1, n2

ret

compilación

```
Scn emulator screen (80x25 chars)
Dame el primer valor: 3
Dame el segundo valor 2
resta = 1
```

Ejercicio 3 Buenos días

FIN MACRO

MOV AH, 4CH ;TERMINACION DEL PROGRAMA

INT 21H

ENDM

;-----

PILA SEGMENT PARA STACK "STACK"

DB 200 DUP (0) ;ESPACIO DE LAS INSTRUCCIONES

PILA ENDS

;-----

DATOS SEGMENT PARA "DATA" ;DATOS A INGRESAR

VA DB "DESPUES DE MEDIO DIA (Y/N): ", "\$" ;SELECCION DE INICIO DEL PROGRAMA

VD DB 13,10," BUENOS DIAS", "\$" ;MENSAJES MOSTRADOS EN PANTALLA

VT DB 13,10," BUENAS TARDES", "\$" ;MENSAJES MOSTRADOS EN PANTALLA

DATOS ENDS

;-----

CODIGO SEGMENT PARA "CODE"

EMPIEZA PROC FAR

ASSUME CS: CODIGO, DS: DATOS, SS: PILA

MOV AX, DATOS ;MANDAR LLAMAR A DATOS

MOV DS, AX

MOV AH, 0FH

INT 10H

MOV AH, 00H

INT 10H

MOV DX,OFFSET VA ;IMPRIMIR MENSAJE "DESPUES DE MEDIO DIA"

MOV AH,9

INT 21H

CALL PREGUNTA

EMPIEZA ENDP

PREGUNTA PROC NEAR ;Inicia el

MOV AH, 01H ;ESPERA UNA ACCION

INT 21H

CMP AL, "Y" ;Asignar una accion a una tecla

JZ TARDES

CMP AL, "N" ;Asignar una accion a una tecla

JZ DIAS

CMP AL, "y" ;Asignar una accion a una tecla

JZ TARDES

CMP AL, "n" ;Asignar una accion a una tecla

JZ DIAS

TARDES:

MOV DX, OFFSET VT ;IMPRIMIR MENSAJE "BUENOS TARDES"

MOV AH, 09

INT 21H

FIN

DIAS:

MOV DX, OFFSET VD ;IMPRIMIR MENSAJE "BUENOS DIAS"

MOV AH, 09

INT 21H

FIN

RET

PREGUNTA ENDP ;FIN DE PREGUNTA PROC

CODIGO ENDS

END EMPIEZA ;FIN DEL PROGRAMA

SCR emulator screen (80x25 chars)

```
DESPUES DE MEDIO DIA (Y/N): Y
BUENAS TARDES
```

Ejercicio 4

imprime macro numero ;Nuestra macro se llama imprimir, nuestro parámetro es numero

```
mov ah,02h
```

```
mov dl,numero ;Indica que mueva al registro DL lo que pasamos como parámetro.
```

```
add dl,30h ;suma 30h para imprimir el número real.
```

```
int 21h
```

```
endm
```

lup macro

```
mov num,cl
```

```
imprime num ;Llamamos al macro con el valor de nuestra variable.
```

```
loop inicio ;repite ciclo
```

```
endm
```

Fin macro

```
mov ah,04ch ;Finaliza el programa.
```

```
int 21h
```

```

endm

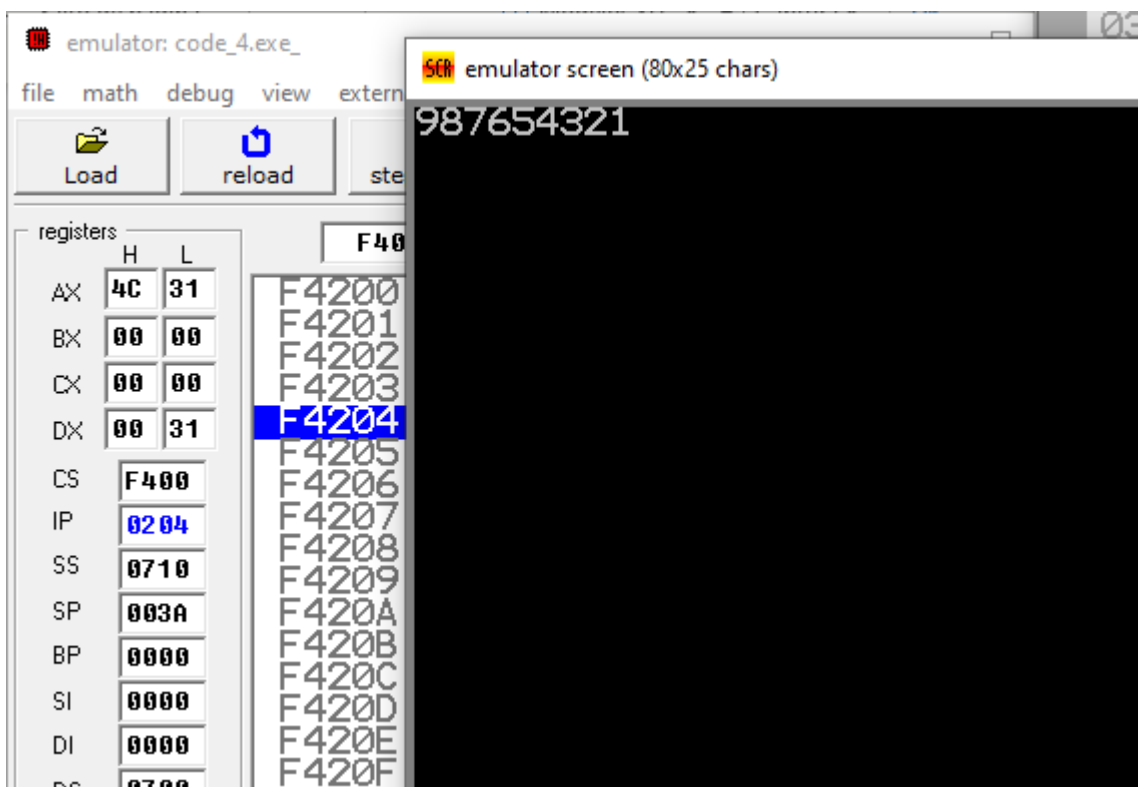
.model small
.stack 64
.data
    num db 0 ;declaramos nuestra variable.
.code
    mov cx,9
inicio:

    lup

    Fin

End

```



Prácticas de Macros Externas

Practica 1

imprime macro numero ;Nuestra macro se llama imprimir, nuestro parámetro es numero

```
mov ah,02h
```

```
mov dl,numero ;Indica que mueva al registro DL lo que pasamos como parámetro.
```

```
add dl,30h ;suma 30h para imprimir el número real.
```

```
int 21h
```

```
endm
```

```
.model small
```

```
.stack 64
```

```
.data
```

```
num db 0 ;declaramos nuestra variable.
```

```
.code
```

```
mov cx,9
```

```
inicio:
```

```
mov num,cl
```

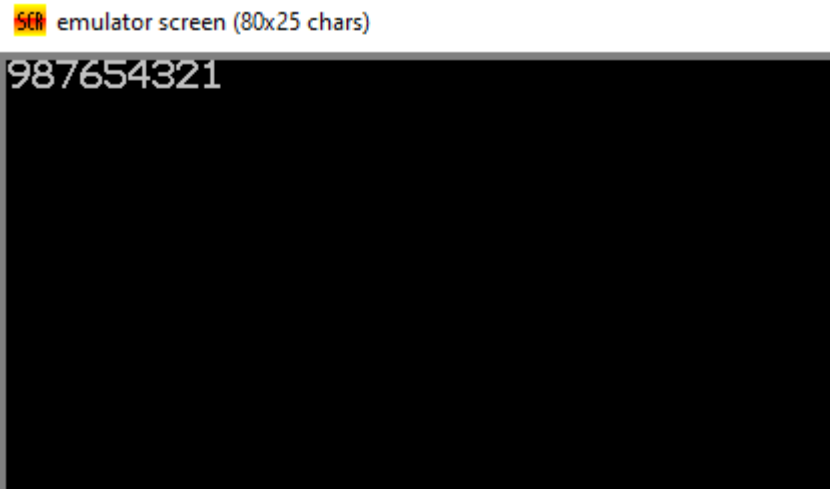
```
imprime num ;Llamamos al macro con el valor de nuestra variable.
```

```
loop inicio ;repite ciclo
```

```
mov ah,04ch ;Finaliza el programa.
```

```
int 21h
```

```
end
```

 emulator screen (80x25 chars)

987654321

```
include 'emu8086.inc'
```

```
include macros.asm
```

```
.data
```

```
mensaje db "Programa de macro que imprime del 9 al 0", 13, 10, "$"
```

```
mensaje2 db 13, 10, "Loop que imprime del 0 al 9", 13, 10, "$"
```

```
num db 0
```

```
.code
```

```
mov cx, 10
```

```
lea dx, mensaje
```

```
mov ah, 09h
```

```
int 21h
```

```
mov cl, 9
```

```
imprimir_numeros:
```

```
mov dl, cl
```

```
add dl, '0'
```

```
mov ah, 02h
```

int 21h

loop imprimir_numeros

lea dx, mensaje2

mov ah, 09h

int 21h

mov cl, 0

imprimir_numeros2:

mov dl, cl

add dl, '0'

mov ah, 02h

int 21h

inc cl

cmp cl, 10

jl imprimir_numeros2

mov ah, 4Ch

int 21h

ret

PRINT_NUM_LOOP:

```

xor dx, dx
div cx
add dl, '0'
push dx
inc bx
test ax, ax
jnz PRINT_NUM_LOOP

```

PRINT_NUM_DISPLAY_LOOP:

```

pop dx
mov ah, 02
int 21h
dec bx
jnz PRINT_NUM_DISPLAY_LOOP

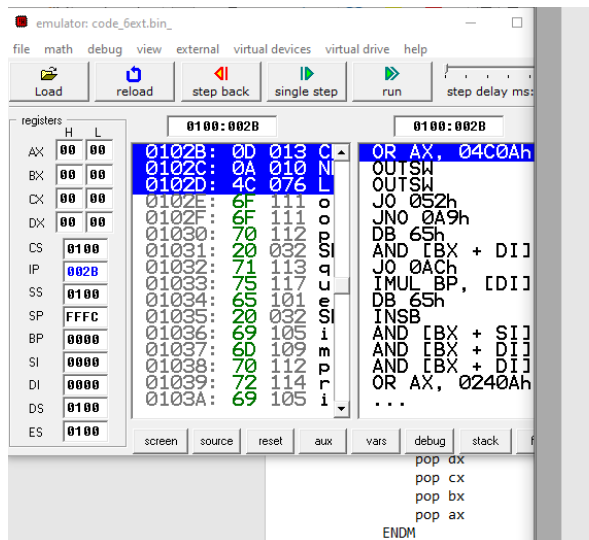
```

```

pop dx
pop cx
pop bx
pop ax

```

ENDM





Números del 0 al 9:
0123456789

SUMAS

PRACTICA 3

include macros.asm

.model small

.stack 64

.data

n1 db 0

n2 db 0

suma db 0

msj db "Dame el primer valor: \$"

msj2 db 10, 13, "Dame el segundo valor \$"

msj3 db 10, 13, "suma = \$"

.code

mov ax, @data

mov ds, ax

mov ah, 09

lea dx, msj

int 21h

mov ah, 01

int 21h

sub al, 30h

mov n1, al

mov ah, 09

lea dx, msj2

int 21h

mov ah, 01

int 21h

sub al, 30h

mov n2, al

sumas n1, n2

ret

Scn emulator screen (80x25 chars)

```
Dame el primer valor: 2
Dame el segundo valor 3
```

```
La suma de 2 + 3 es: 5
```

macros: Bloc de notas

Archivo Edición Formato Ver Ayuda

; macros.asm

.model small

.data

suma db 0

msj3 db 10, 13, "suma = \$"

.code

sumas proc

; Procedimiento para sumar n1 y n2

; Entrada: n1, n2

; Salida: suma

mov al, n1 ; Carga n1 en AL

add al, n2 ; Suma n2 a AL

mov suma, al ; Guarda el resultado en suma

mov ah, 02 ; Función para imprimir un carácter

mov dl, suma ; Carga el valor de suma en DL

add dl, 30h ; Convierte DL a carácter

int 21h

ret

sumas endp

Practica 4

include 'macros.txt'

data segment

msg DB "Resultado: \$"

var dw 6

ends

```
main proc
mov ax,@data
    mov ds, ax
    mov dx,offset msg
    mov ah,09h
    int 21h
```

```
mov bx,1
mov ax,1
m_suma bx
```

```
m_final
```

```
end main
```


contenido del archivo .txt

```
m_final macro
mov ah, 4ch
int 21h
endm
```

```
m_suma macro var
add ax, bx
mov ah,02
mov dx,ax
add dx,30h
```



int 21h

endm

 emulator screen (80x25 chars)



Resultado: 2

 macros: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
m_final macro
    mov ah, 4ch
    int 21h
endm

m_suma macro var
    add ax, bx
    mov ah, 02
    mov dx, ax
    add dx, 30h
    int 21h
endm
|
```

Practica de Macros internas y Procedimiento

Practica 1

; multi-segment executable file template.

data segment

```
; add your data here!  
pkey db "ejercicio de macro con procedimiento...$"  
texto db "Este es otro$"  
ends
```

```
stack segment  
    dw 128 dup(0)  
ends
```

```
print macro ren,col,cad  
    mov ch, ren  
    mov cl,col  
    push cx  
    lea dx,cad
```

```
    push dx  
    call imprime  
    pop dx  
    pop ax  
endm
```

```
code segment  
;  
mover proc  
    pusha  
    mov bp,sp  
;  
    mov dx,[bp+18]  
    mov ah,2
```

```
mov bx,0
int 10h
popa
ret
endp
```

```
;
```

```
imprime proc
```

```
pusha
mov bp,sp
```

```
mov ax,[bp+20]
push ax
call mover
pop ax
```

```
mov dx,[bp+18]
```

```
;
```

```
mov ah, 9
int 21h
popa
```

```
ret
```

```
endp
```

```
start:
```

```
;
```

```
mov ax, data
mov ds, ax
mov es, ax
```

```
print 3,20,pkey
```

```
print 10,10,texto
```

```
; wait for any key....
```

```
mov ah, 1
```

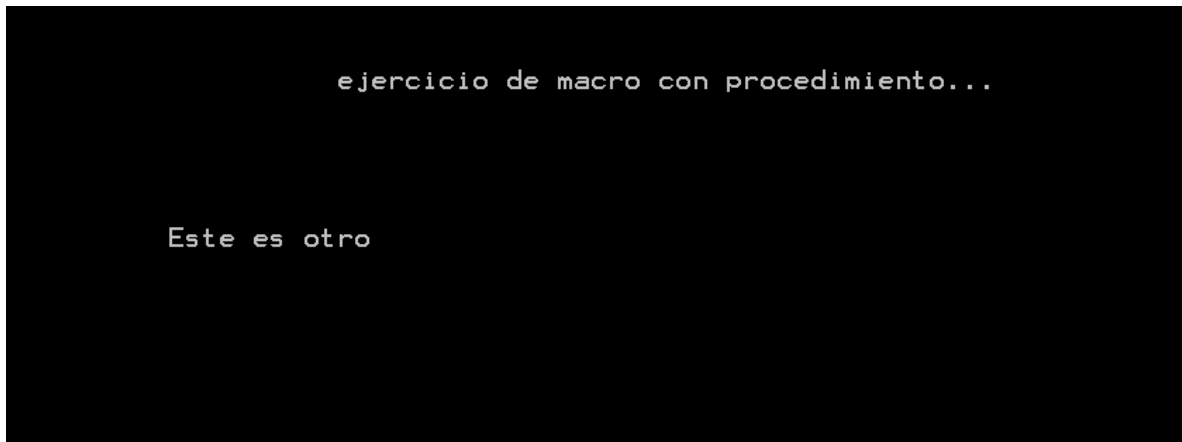
```
int 21h
```

```
mov ax, 4c00h ; exit to operating system.
```

```
int 21h
```

```
ends
```

```
end start ; set entry point and stop the assembler.
```



Practica 2

```
include 'emu8086.inc'
```

```
org 100h
```

```
mov si, 0
```

```
printn "La Suma es:"
```

```
sumar proc
```

```
    mov al, [matriz + si]
    mov bl, [matriz2 + si]
    add al, bl
    printn ""
    call print_num
    printn ""
    inc si
    cmp si, 9
    jne sumar
    jmp fin
    ret
sumar endp
```

```
fin:
    jmp opcion
```

```
opcion proc
    printn "Opción elegida"
    ret
opcion endp
```

```
define_clear_screen
define_pthis
define_scan_num
define_print_num
define_print_num_uns
```

```
matriz db 9, 8, 7, 6, 5, 4, 3, 2, 1
matriz2 db 1, 2, 3, 4, 5, 6, 7, 8, 9
```

568 emulator screen (80x25 chars)

```
La Suma es:
10
10
10
10
10
10
10
10
10
10
10
Opcion elegida
```

Practica 3

```
include 'emu8086.inc'
```

```
org 100h
```

```
jmp inicio
```

```
M1 dw 2 dup (?, ?)
```

```
M2 dw 2 dup (?, ?)
```

```
M3 dw 2 dup (?, ?)
```

```
numx dw ?
```

```
numy dw ?
```

```
; Define macros
```

```
macro CapturaValor registro
```

```
push ax
println ''
println ''
print ' ', registro, '['
mov ax, numx
call print_num
print ' , '
mov ax, numy
call print_num
print ' ] ----> '
call scan_num
mov registro, cx
pop ax
endm
```

```
macro ImprimeResultado
    println ''
    println ' Imprimir el resultado de la resta '
    println ''
endm
```

```
macro ImprimeValor registro
    push ax
    println ''
    println ''
    print ' ', registro, '['
    mov ax, numx
    call print_num
    print ' , '
```

```
    mov ax, numy
    call print_num
    print ' ] ----> '
    mov ax, registro
    call print_num
    pop ax
endm
```

; Define procedimientos

```
proc CapturaCeldas
```

```
    push si
    push ax
    mov si, 0
    mov numy, 0
```

```
Captura_celdas_loop:
```

```
    CapturaValor M1[si]
    CapturaValor M2[si]
```

```
    xor ax, ax
    xor bx, bx
    mov ax, M1[si]
    mov bx, M2[si]
    sub ax, bx
    mov M3[si], ax
```

```
    inc numy
    add si, 2
```



```
    mov ax, numy  
    cmp ax, 2  
    jb Captura_celdas_loop
```

```
    pop ax  
    pop si  
endp
```

```
proc ImprimirCeldas
```

```
    push si  
    push ax  
    mov si, 0  
    mov numy, 0
```

```
Imprimir_celdas_loop:
```

```
    ImprimeValor M3[si]
```

```
    inc numy  
    add si, 2
```

```
    mov ax, numy  
    cmp ax, 2  
    jb Imprimir_celdas_loop
```

```
    pop ax  
    pop si  
endp
```

```
inicio:
```

```
mov cx, 2
mov si, 0
mov numx, 0
mov numy, 0
```

Captura_filas:

```
push cx
mov numy, 0
```

Captura_celdas:

```
CapturaValor M1[si]
CapturaValor M2[si]
```

```
call CapturaCeldas
```

```
inc numx
pop cx
loop Captura_filas
```

```
mov cx, 2
mov si, 0
mov numx, 0
mov numy, 0
```

ImprimeResultado:

Imprimir_resultado:

```
push cx
mov numy, 0
```

Imprimir_celdas:

call ImprimirCeldas

inc numx

pop cx

loop Imprimir_resultado

ret

define_clear_screen

define_pthis

define_scan_num

define_print_num

define_print_num_uns

end

568 emulator screen (80x25 chars)

```
0 , 0 1 ----> 2
0 , 0 1 ----> 2
0 , 0 1 ----> 1
0 , 0 1 ----> 1
0 , 1 1 ----> 2
0 , 1 1 ----> 2
0 , 0 1 ----> 0
0 , 1 1 ----> 0
0 , 0 1 ----> 0
0 , 0 1 ----> 0
```

Practica 4

```
include 'emu8086.inc'
```

```
org 100h
```

```
; Salta a la etiqueta "inicio"
```

```
jmp inicio
```

```
; Define las matrices M1, M2 y M3
```

```
M1 dw 2 dup (?, ?)
```

```
M2 dw 2 dup (?, ?)
```

```
M3 dw 2 dup (?, ?)
```

```
; Define las variables numx y numy
```

```
numx dw ?
```

numy dw ?

; Define macros

macro CapturaValor registro

push ax

println ''

println ''

print ' ', registro, '['

mov ax, numx

call print_num

print ' , '

mov ax, numy

call print_num

print '] ----> '

call scan_num

mov registro, cx

pop ax

endm

macro ImprimeResultado

println ''

println ' Imprimir el resultado de la multiplicación '

println ''

endm

macro ImprimeValor registro

push ax

println ''

println ''

```

    print ' ', registro, '['
    mov ax, numx
    call print_num
    print ' , '
    mov ax, numy
    call print_num
    print ' ] ----> '
    mov ax, registro
    call print_num
    pop ax
endm

```

; Define procedimiento para capturar las celdas

```
proc CapturaCeldas
```

```

    push si
    push ax
    mov si, 0
    mov numy, 0

```

```
Captura_celdas_loop:
```

```

    CapturaValor M1[si]
    CapturaValor M2[si]

```

```

    xor ax, ax
    xor bx, bx
    mov ax, M1[si]
    mov bx, M2[si]

```

; Realiza la multiplicación en lugar de la resta

mul bx

; Almacena el resultado de la multiplicación en M3

mov M3[si], ax

inc numy

add si, 2

mov ax, numy

cmp ax, 2

jb Captura_celdas_loop

pop ax

pop si

endp

; Define procedimiento para imprimir las celdas

proc ImprimirCeldas

push si

push ax

mov si, 0

mov numy, 0

Imprimir_celdas_loop:

ImprimeValor M3[si]

inc numy

add si, 2

```
mov ax, numy  
cmp ax, 2  
jb Imprimir_celdas_loop
```

```
pop ax  
pop si  
endp
```

```
inicio:  
mov cx, 2  
mov si, 0  
mov numx, 0  
mov numy, 0
```

```
Captura_filas:  
push cx  
mov numy, 0
```

```
Captura_celdas:  
CapturaValor M1[si]  
CapturaValor M2[si]
```

```
call CapturaCeldas
```

```
inc numx  
pop cx  
loop Captura_filas
```

```
mov cx, 2
```


mov si, 0

mov numx, 0

mov numy, 0

ImprimeResultado:

Imprimir_resultado:

push cx

mov numy, 0

Imprimir_celdas:

call ImprimirCeldas

inc numx

pop cx

loop Imprimir_resultado

ret

define_clear_screen

define_pthis

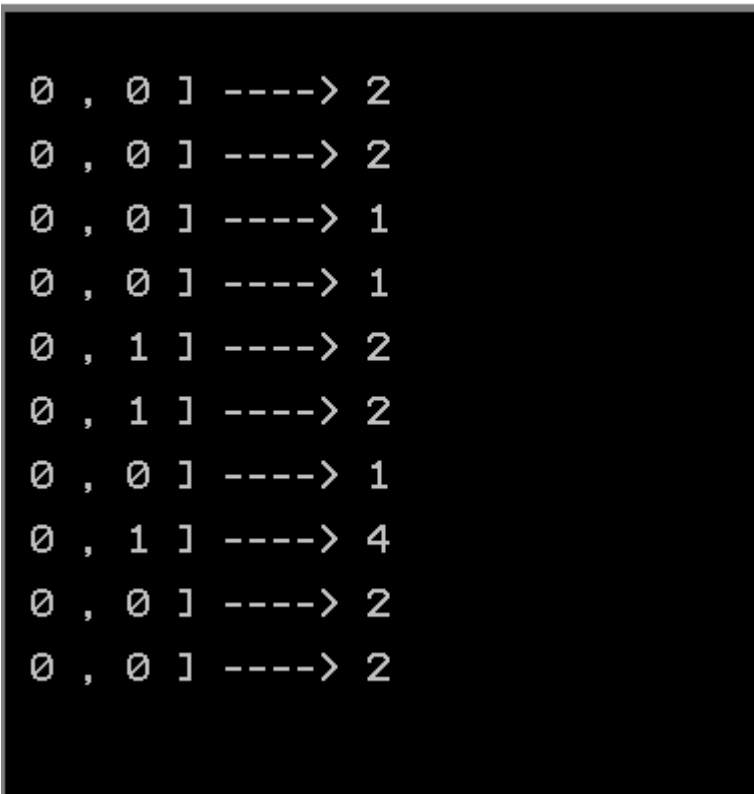
define_scan_num

define_print_num

define_print_num_uns

end

Stfl emulator screen (80x25 chars)



Macros Externas y Procedimiento

```
include "emu8086.inc"
```

```
include "Operaciones.txt"
```

```
org 100h
```

```
matriz db 9 dup (0)
```

```
matriz2 db 9 dup (0)
```

```
mov si,0
```

```
printn 'ingrese los primero numeros de tu primera matriz'
```

```
call principal
```

```
principal proc
```

```
    call Mtz1
```

```
    call Mtz2
```

```
    call opcion
```

```
    call macro_suma
```

```
    call macro_resta
```

```
    call macro_multi
```

```
    call macro_divi
```

```
    ret
```

```
principal endp
```

```
Mtz1 proc
```

```
    call scan_num
```

```
    printn "
```

```
    mov matriz[si],cl
```

```
    inc si
```

```
    cmp si,9
```

```
    jne Mtz1
```

```
    mov si,1
```

```
    mov bx,0
```

Mtz1 endp

mov si,0

print "ingresa los numeros de la segunda matriz:"

Mtz2 proc

call scan_num

printn "

mov matriz2[si],cl

inc si

cmp si,9

jne Mtz2

mov si,1

mov bx,0

Mtz2 endp

opcion proc

proc "====menu===="

printn ""

printn "seleccion opcion"

printn "1 suma"

printn "2 resta"

printn "3 multiplicar"

printn "4 dividir"

```
printn "5 salir"
```

```
printn "=====
```

```
call scan_num
```

```
printn "
```

```
mov bx,cx
```

```
cmp bx,1
```

```
je macro_suma
```

```
cmp bx,2
```

```
je macro_resta
```

```
cmp bx,3
```

```
je macro_multi
```

```
cmp bx,4
```

```
je macro_divi
```

```
cmp bx,5
```

```
printn "saliendo"
```

```
je call salir
```

```
jmp opcion
```

```
opcion endp
```

```
macro_suma proc
```

```
suma
```

```
macro_resta proc
```

```
resta
```

macro_multi proc

multiplicacion

macro_divi proc

dividision

salir proc

print 'precione una tecla para salir'

mov ah,0h

int 16h

salir endp

define_clear_screen

define_pthis

define_scan_num

define_print_num

define_print_num_uns

end

```
suma macro
mov si,0
printn "La Suma es:"
    sumar proc
        mov al,matrix[si]
        mov bl,matrix2[si]
        add al,bl
        printn""
        call print_num
        printn""
        inc si
        cmp si,9
        jne sumar
        jmp call opcion
        ret
    sumar endp
endm
```

```
resta macro
mov si,0
printn "La Resta es:"
```

```
    restar proc
        mov al,matrix[si]
        mov bl,matrix2[si]
        sub al,bl
        printn" "
        call print_num
        printn" "
        inc si
        cmp si,9
        jne restar
        jmp call opcion
        ret
    restar endp
endm
```

```
emu8086 emulator screen (80x25 chars)
27
30
9
16
20
Ingresa los numeros de la segunda matriz:
5
10
7
11
9
6
3
8
20
=====Menu=====
Selecciona Opcion a Realizar
1.-Suma
2.-Resta
3.-Multiplicacion
4.-Divicion
5.-Salir
=====
1
```

Practica 2

```
include 'emu8086.inc'
```

```
include 'macros.inc'
```

```
.model small
```

```
data segment
```

```
    base dw ?
```

```
    exp  dw ?
```

```
    resul dw ?
```

```
ends
```

```
code segment
```

```
call basic
```


basic proc

call start

call objective

call capture

potencia_num

call printer

call salir

ret

basic endp

start proc

mov ax, data

mov ds, ax

mov es, ax

ret

start endp

mov ax,data

mov ds,ax

mov es, ax

ret

objective endp

capture proc

printn ' '

printn ' '

printn "proporciona un numero =>"

call scan_num

```

mov base,cx

    printn ' '
    printn ' '
    printn "Proporciona el exponente"
    call scan_nummov exp,cx
    ret
capture endp

printer proc

    printn ' '
    printn ' '
    printn "Resultado"
    call print_num
    ret
printer proc

salir proc

    printn ' '
    printn ' '
    printn 'precione una tecla para continuar...'
    mov ah, 1
    int 21h
    mov ax, 4c00h
    int 21h
    ret
salir endp

```

define_scan_num

define_print_string

define_print_num

define_print_num_uns

define_phis

ends



macros: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
m_final macro
    mov ah, 4ch
    int 21h
endm

m_suma macro var
    add ax, bx
    mov ah, 02
    mov dx, ax
    add dx, 30h
    int 21h
endm
```

Potencia de un Numero

Dame tu numero ==> 6

dame el exponente ==> 3

Resultado ==> 216

