ARQUITECTURA DE COMPUTADORAS

CICLO LECTIVO: 2021

ASSEMBLER 8086

EJERCICIOS RESUELTOS

Ejercicios Propuestos

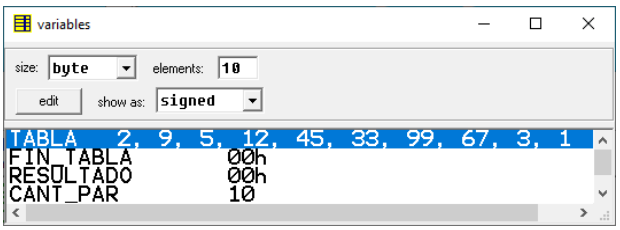
1.-Realice un programa en assembler que intercambie los contenidos de dos valores decimales definidos en dato1 y dato2.

```
; Ejercicio 1: intercambie los contenidos de dos
; valores decimales definidos en dato1 y dato2.
;
; XCHG realiza el intercambio entre los valores de los
; operandos de la siguiente forma:
;
; XCHG reg, mem
; XCHG reg, reg
; XCHG mem, reg

ORG 100h

dato1 DB 0
dato2 db 0

mov dato1, 45
mov dato2, 60

mov al, dato1
mov ah, dato2
XCHG al, al

mov dato1, ah
mov dato2, al
```

2.- Realice un programa assembler que cuente la cantidad de números pares, definidos por el siguiente segmento de datos: 2,9,5,12,45,33,99,67,3,1. El resultado deberá almacenarse en la variable cant_par.

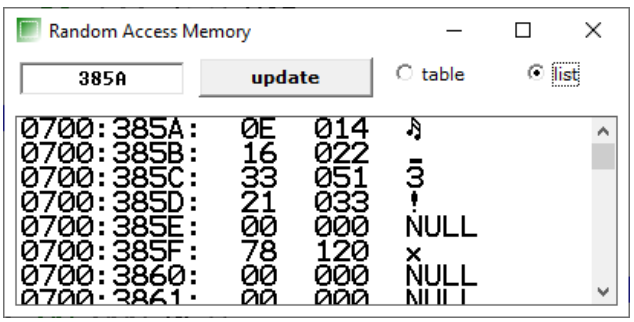
```
; Ejercicio Nro 2: contar cantidad de nros.pares
; segmento de datos: 2,9,5,12,45,33,99,67,3,1.
; Resultado en variable cant_par
;
; AND entre el elemento leido en AL (binario)
; y mascara 0000 0001 devuelve 0 (par) o
; 1 (impar).
;
; JZ Salto si resultado es cero (cant_par)
; sino cuento impar (cant_impar)

ORG 100h

tabla DB 2,9,5,12,45,33,99,67,3,1

fin_tabla DB ?
resultado DB 0
cant_par  db 0

MOV BX, OFFSET tabla                        ; Carga en BX direccion de Tabla
MOV CL, OFFSET fin_tabla - OFFSET tabla     ; Carga en Cl la canidad de elementos

Loop:
```

```
      MOV AL, [BX]                    ; [BX] contenido direccion de BX
      INC BX
      AND AL ,1B
      JZ  PAR                         ; Salta a PAR: si resultado es = 0
S:    DEC CL                          ; decrementa CL contador loops
      JNZ Loop                        ; Itera si resultado es <>0
      HLT

PAR:  INC cant_par
      JMP S
```



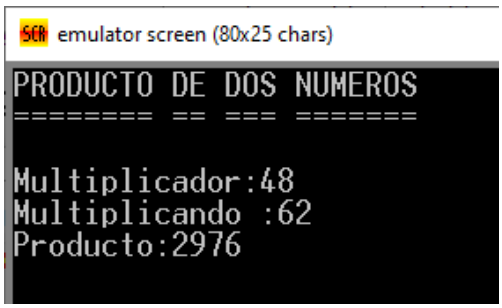
3- Realice un programa Assembler que permita sumar el contenido de las siguientes direcciones 2Ah, 2Bh, 2Ch y 2Dh guardando el resultado en 0Dh. Cargar dichas direcciones previamente con valores de operandos.

```
; Ejercicio 3: sumar contenido de las direcciones
; 385Ah, 385Bh, 385Ch y 385Dh.
; inicializadas con valores 14, 22, 51, 33
; Guardar resultado en 385Fh.
;
; [DIR] = contenido de DIR


ORG 100h

mov [385Ah],14      ; mueve 14 a dir 385Ah
mov [385Bh],22      ; mueve 22 a dir 385Bh
mov [385Ch],51      ; mueve 51 a dir 385Ch
mov [385Dh],33      ; mueve 33 a dir 385Dh

ADD AL,[385Ah]      ; Al = Al + (385Ah)
ADD AL,[385Bh]      ; Al = Al + (385Bh)
ADD AL,[385Ch]      ; Al = Al + (385Ch)
ADD AL,[385Dh]      ; Al = Al + (385Dh)

mov [385Fh],AL      ; mueve (AL) a Dir 385Fh
```



4- Construya un programa en assembler que realice la multiplicación de dos números imprimiendo el resultado en pantalla.

```
include 'emu8086.inc'

ORG    100h
producto    dw 0

;imprime titulos

PRINTN 'PRODUCTO DE DOS NUMEROS'
PRINTN '======== == === ======='

printn                    ; salto de linea
print 'Multiplicador:'
CALL   scan_num           ; pide numero y almacena en CX.
ADD    producto, cx

printn
print 'Multiplicando :'
CALL   scan_num           ; multiplicando en CX.

multi:                    ; loop hasta que cx=0
    add ax,producto
    loop multi

printn
print 'Producto:'
CALL   print_num          ; imprime varlor de AX.

RET


;--- definicion de procedimientos de impresion
;--- y captura de datos de libreria emu8086.inc

DEFINE_SCAN_NUM

DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; definir si se usa print_num.
END
```



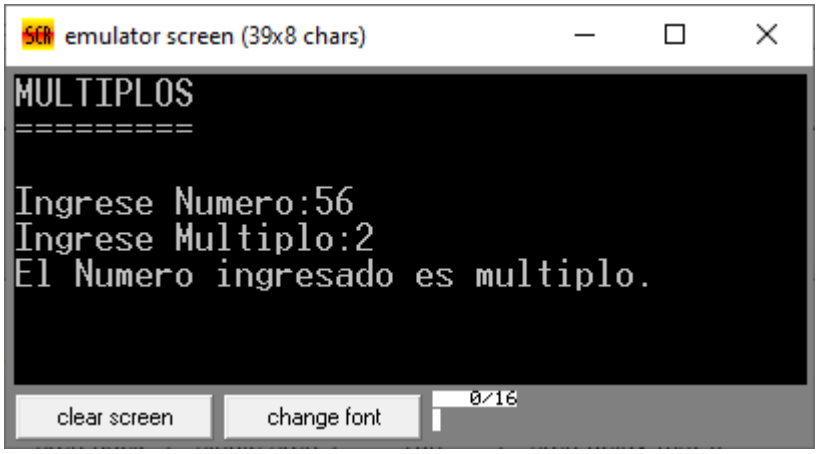

```
; Ejercicio 4.
; realice la multiplicacion de dos numeros ingresados por teclado
; imprima resultado por pantalla y guarde en variable result
```

```
include 'emu8086.inc'
name "multiplicacion"
.MODEL SMALL
org 100h
.DATA
; VARIABLES
    op1 db 0
    op2 db 0
    result db 0
    msg0 db 0Dh,0Ah,"MULTIPLICACION DE DOS NUMEROS $"
    msg1 db 0Dh,0Ah,"ingrese numero 1: $"
    msg2 db 0Dh,0Ah,"ingrese numero 2: $"
    msg3 db 0Dh,0Ah,"La multiplicacion es : $"

.CODE
    mov dx, offset msg0  ; IMPRIME TITULOS
    mov ah, 9
    int 21h

    mov dx, offset msg1  ; IMPRIME LEER VARLOR 1
    mov ah, 9
    int 21h

    CALL   scan_num            ; pide numero y almacena en CX.
    MOV    op1, CX          ; ALMACENA NUMERO EN OP1

    mov dx, offset msg2       ; IMPRIME LEER VARLOR 2
    mov ah, 9
    int 21h

    CALL   scan_num            ; pide numero y almacena en CX.
    MOV    op2, CX          ; ALMACENA NUMERO EN OP2


    mov ax, 0           ; PONE A CERO AX

multi:
    add     al,op1                ; SUMA VALOR
    loop    multi             ; LOOP Y DECREMENTA CX

    mov    result, ax           ; guarda resultado
    push    ax                ; resguardo valor
    mov    dx, offset msg3        ; IMPRIME MENSAJE DE RESULTADO
    mov ah, 9
    int 21h
    pop    ax           ; RECUPERO VALOR
    CALL   print_num           ; imprime valor de AX.
    RET                        ; vuelve al sistema operativo.

;--- definicion de procedimientos de impresion

;--- y captura de datos de libreria emu8086.inc

DEFINE_SCAN_NUM

DEFINE_PRINT_STRING

DEFINE_PRINT_NUM

DEFINE_PRINT_NUM_UNS  ; definir si se usa print_num.
```

```
DEFINE_PTHIS
END
```

5- Realice un programa Assembler que verifique si un número ingresado es múltiplo de otro.

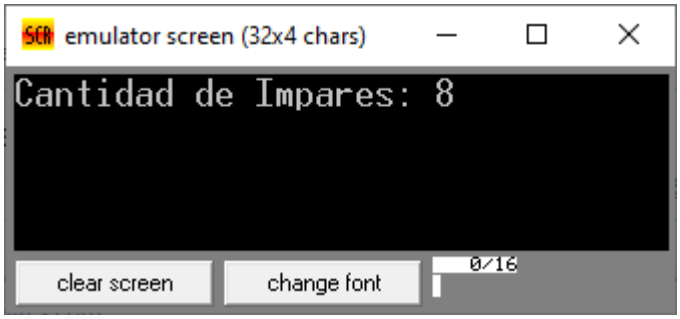
```
include 'emu8086.inc'

; DIV: (byte)
;        AL = AX / operando
;        AH = resto

ORG    100h

;imprime titulos
printn 'MULTIPLOS'
printn '========='
printn                  ; salto de linea
print 'Ingrese Numero:'
call scan_num           ; input en CX.
mov ax,cx               ; mueve dividendo a ax
printn
print 'Ingrese Multiplo:'
call scan_num           ; input en CX.
printn
mov bl,cx               ; mueve divisor a bl
div bl                  ; divide (ax) / bl y ah=resto
cmp ah,0                ; compara resto en ah con 0

je nm
   PRINT 'El Numero ingresado no es multiplo.'
   jmp exit

nm:
   PRINT 'El Numero ingresado es multiplo.'
exit:
   hlt

;--- definicion de procedimientos de impresion
;--- y captura de datos de libreria emu8086.inc

DEFINE_SCAN_NUM

DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; definir si se usa print_num.
END
```



6.- Realice un programa en assembler que recorra el siguiente segmento de datos: 2,9,5,12,45,33,99,67,3,1. Deberá mostrar en pantalla la cantidad de números impares.

```
include 'emu8086.inc'
;
;Ejercicio Nro. 6: recorra segmento 2,9,5,12,45,33,99,67,3,1
; y contar cantidad de numeros impares (no divisibles por 2)
;
; AND entre valor binario del elemento y una mascara 0001
; devuelve 0-> PAR y 1-> Impar
;
; JZ: Salto si
ORG 100h
tabla DB 2,9,5,12,45,33,99,67,3,1
fin_tabla DB ?
cant_impar  DB 0

MOV BX, OFFSET tabla
; Cantidad elementos a CL
MOV CL, OFFSET fin_tabla - OFFSET tabla

Loop: MOV Ax, [BX]
      INC BX
      ; elemento and mascara=1
      AND Ax ,1
      ; Salto a contar Impar si no es Cero
      JNZ IMPAR
S:    DEC CL
      JNZ Loop

      ;imprime resultado
      mov al, cant_impar
      print 'Cantidad de Impares: '
      CALL   print_num; imprime varlor de AX.

      HLT

IMPAR:INC cant_impar
      JMP S

;--- definicion de procedimientos de impresion
;--- y captura de datos de libreria emu8086.inc

DEFINE_SCAN_NUM
```

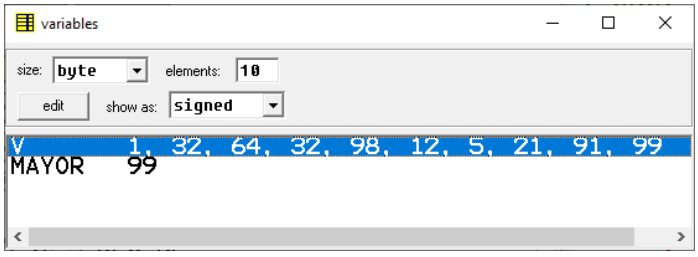
```
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; definir si se usa print_num.
END
```




```
;Ejercicio 6. Recorra el vector y cuente la cantidad de valores IMPARES
; imprime resultado por pantalla.
; suma  vector y guarda resultado en Ax
name "suma-impares"
.MODEL SMALL
org 100h
.DATA
    vec db 10,21,2,200,35,44,21,9,11,22 ; cargar un vector con constantes
    mensaje db "cantidad de impares: $" ;
.CODE
    ;mov ax,@Data
    ;mov ds, ax
    mov bx,0    ; en bx tenemos la posición dentro del vector, inicia en 0
    mov cx,10   ; descontador cantidad de posiciones del vector.
    mov ax,0    ; suma
    mov dx,0
bucle:
    mov dl,vec[bx] ; cargamos en DL el elemen0to del vector indicado en BX
    inc bx         ; incrementamos BX, avanza el vector
    and dl,1       ; AND logica con literal 1 para determinar valor de bit de
menor peso
    jz esPar       ; salto condicional si resultado anterior fue cero, es par.
    add ax,1       ; sino salta es impar , incremento en uno acumulador de
impares
esPar:
    loop bucle     ; loop y decrementa cx.
                   ; fin del loop , imprimir resultado.
    push ax        ;resguardamos en pila el resultado
    mov dx,offset mensaje          ; cargo mensaje a imprimir
    mov ah,09h                     ; valor a cargar para imprimir caracteres
    int 21h                        ; llamada a interrupcion.
    pop ax          ; recupero valor
    mov dx,ax       ; muevo a datos para imprimir
    add dl,30h      ; sumo para representar el numero como caracter ascii
    mov ah,02h      ;valor a cargar para imprimir numeros
    int 21h         ; llamada a interrupcion
    ret             ; volver al sistema operativo
```

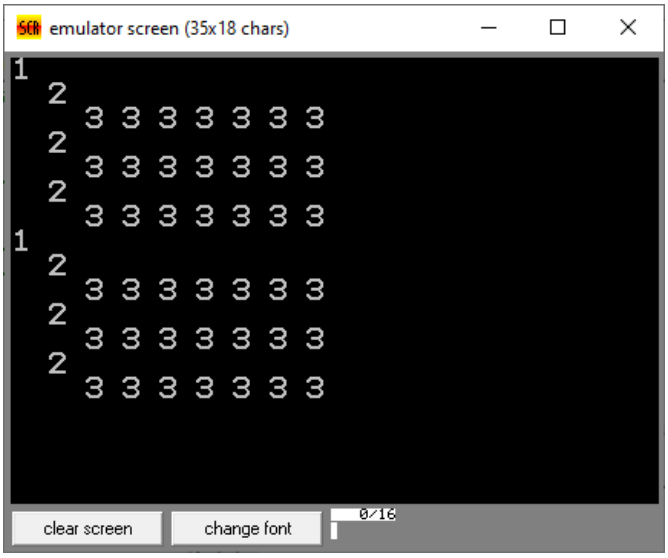
7- Realice un programa Assembler que recorra el siguiente vector: 1,32,64,32,98,12,5,21,91,99. Deberá almacenar en variable mayor, el valor mayor del vector

```
; Ejercicio Nro 3: recorra vector 1,32,64,32,98,12,5,21,91,99.
; y almacene su mayor valor en variable Mayor
;
; CX: contador que evoluciona en cada iteracion Loop
;
; CMP:COmpara dos operandos y setea flags(OF, SF, ZF, AF, PF, CF)
; de acuerdo al resultado de la comparacion
;
; JNG: Salto si primer operando no es mayor que el segundo operando
; (segun orden de comparacion de CMP

org 100h
    mov cx,9
inicio:
    mov si, cx
    mov dl, v[si]
    cmp dl, mayor
    jng siguiente ; Short salta si operando1<= operando2
    mov mayor, dl
siguiente:
    loop inicio
    mov al, mayor
   ret

   v db 1,32,64,32,98,12,5,21,91,99
   mayor db 0
```



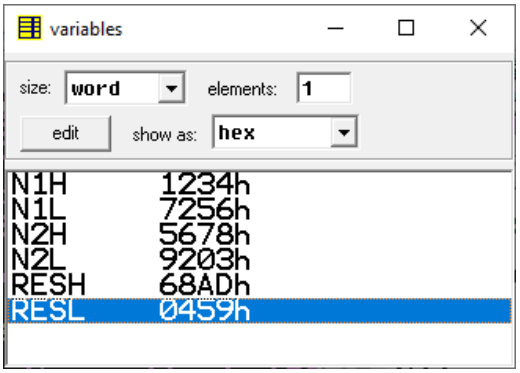
8- Realice un programa que anide 3 bucles. Se deberá poder definir la cantidad de iteraciones de cada bucle, e imprimir en pantalla un indicador numérico de cada iteración.

```
; Programa para realizar 3 bucles anidados
; By Gustavo Maurokefalidis
; Cbucle1, Cbucle2, y Cbucle3 define cantidad
; de iteraciones en cada loop
; con PUSH y POP se guarda en pila valor cx
; antes de entrar a un LOOP nivel+1
; y se lo recupera al regresar a Loop nivel-1

include 'emu8086.inc'
org 100h

CBucle1    dw  2
CBucle2    dw  3
CBucle3    dw  7

mov bx, 0  ; contador de bucles.
mov cx, CBucle1
k1: add bx, 1 ; Bucle #1
    printn '1'
    push cx
    mov cx, CBucle2
      k2: add bx, 1 ; Bucle #2
      printn '  2 '
      push cx
         print '   '
         mov cx, CBucle3
          k3: add bx, 1  ; Bucle #3
              print ' 3'
         loop k3        ; Fin Bucle #3
         pop  cx
      printn
       loop  k2 ; Fin Bucle #3
       pop cx
loop k1 ; Fin Bucle #3

hlt
```



9- Realice un programa que sume dos números hexadecimales de 32 bits previamente definidos.

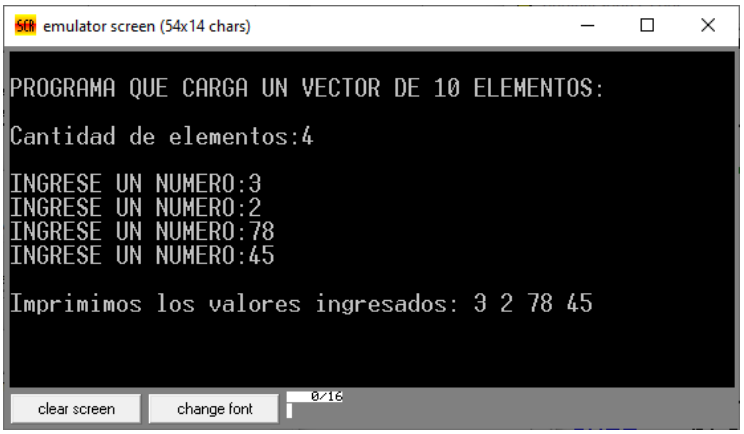
```
; Suma dos numeros de 32 bits
; Gustavo Maurokefalidis
;
; ResH y ResL componen el resultado en 32 bits
; Se define rutina ADD32 para sumar en 32 bits

; ADD32 se la debe definir antes del programa

org 100h

; n1h y n1l -> 12347256h
N1h   dw  1234h
N1l   dw  7256h ; a proposito para que haya carry

; n2h y n2l -> 56789203h
N2h   dw  5678h
N2l   dw  9203h

Resh  dw  ?
Resl  dw  ?

mov ax,N1h
mov bx,N2h
mov cx,N1l
mov dx,N2l

add32:  add cx,dx   ; N1l+N2l
        JNC NoCarry
        inc ax      ; si hay carry pre incremento N1h

NoCarry:add ax,bx   ; N1h+N2h

Result:
        mov ResH,ax ; Bits mayor peso de suma a ax
        mov ResL,cx ; Bits menor peso de suma a cx


hlt
```



10- Realice un programa que permita cargar n elementos en un vector e imprimirlos en pantalla.

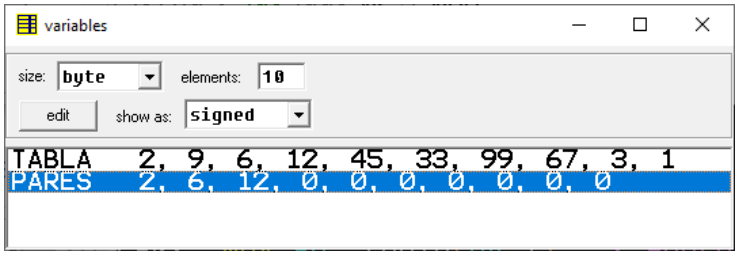
```
include emu8086.inc
name "carga-imprime-vector"
.MODEL SMALL
org 100h
.DATA
    vec  db ? ; cargar un vector con constantes
    msg1 db  13,10,"PROGRAMA QUE CARGA UN VECTOR DE 10 ELEMENTOS:$" ;
    msg2 db  13,10,"Imprimimos los valores ingresados: $" ;
    msg3 db  13,10,"Cantidad de elementos:$>"
    msg  db  13,10,"INGRESE UN NUMERO:$" ;
    cloop   db 0
.CODE
    mov ax, @Data
    mov ds, ax
    mov bx,0            ; posicion vector inicia en 0

    mov dx,offset msg1  ; cargo mensaje a imprimir
    mov ah,09h          ; valor a cargar para imprimir caracteres
    int 21h             ; llamada a interrupcion.
    printn
                        ; imprime linea nueva
    mov dx,offset msg3  ; cargo mensaje a imprimir
    mov ah,09h          ; valor a cargar para imprimir caracteres
    int 21h             ; llamada a interrupcion.
    call scan_num
    mov cloop,cx        ; resguarda cantidad elementos vector
    printn

bucle:
    push cx
    mov dx,offset msg   ; cargo mensaje a imprimir
    mov ah,09h          ; valor a cargar para imprimir caracteres
    int 21h             ; llamada a interrupcion.
    CALL   scan_num     ; pide numero y almacena en CX.
    mov vec[bx], cx     ; cargamos en DL el elemen0to del vector indicado en BX
    inc bx              ; incrementamos BX, avanza el vector
    pop cx
    loop bucle          ; loop y decrementa cx.

    ;imprimimos el vector caargado
    mov bx,0            ; posicion dentro del vector, inicia en 0
    mov cx, cloop       ; cantidad de loops para impresion
    printn
    mov dx,offset msg2  ; cargo mensaje a imprimir
    mov ah,09h          ; valor a cargar para imprimir caracteres
    int 21h             ; llamada a interrupcion.

bucle2:
     push cx
     mov ah, 0
     mov al, vec[bx]
     CALL   print_num   ; imprime varlor de AX.
     inc bx
     pop cx
     print ' '
     loop bucle2

     ret                ; volver al sistema operativo
```

```
;--- definicion de procedimientos de impresion
;--- y captura de datos de libreria emu8086.inc

DEFINE_SCAN_NUM
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; definir si se usa print_num.

END
```



11- Dado un arreglo de 10 numeros, copie a otro solo aquellos que sean pares.

```
; copia pares del arreglo "tabla"
; a nuevo arreglo "pares"
ORG 100h

tabla DB 2,9,6,12,45,33,99,67,3,1
pares DB 10 dup(0)

mov Cx,10   ; indico numero de iteraciones para barrer Tabla
mov Bx, 0   ; primer elemento del array
mov si,0

Bucle:
      mov AL, tabla[BX]     ; [BX] contenido direccion de BX
      inc BX
      and AL ,1B            ; Si AL AND 1 = 0 es par
      jz  PAR               ; Salta a PAR: si resultado es = 0

S:    Loop Bucle            ; decrementa CL contador loops
      HLT

PAR:  mov AH, tabla[BX-1]   ; guardo elemento par en ah
                            ; pero en indice Bx-1 porque ya esta
                            ; preincrementado
      mov pares[si], AH     ; utilizo indice si
      inc si                ;incremento si
      loop S
```

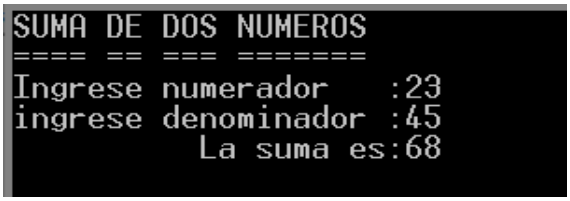


EJERCICIOS COMPLEMENTARIOS

1.- Realice un programa que pida por pantalla dos números, los sume, y presente el resultado. Deberá hacer uso de la librería emu8086.inc y hacer llamadas a las funciones:

```
DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
DEFINE_PTHIS
```

El formato de pantalla deberá ser el siguiente:




```
ORG    100h

;imprime titulos con salto de linea
PRINTN 'SUMA DE DOS NUMEROS'
PRINTN '==== == === ======='

print 'Ingrese numerador :'
CALL   scan_num              ; pide numero y almacena en CX.
ADD    AX, CX                ; suma numero ingresado en CX a AX.
printn                       ; salto de linea

print 'ingrese denominador :'
CALL   scan_num              ; pide numero y almacena en CX.
ADD    AX, CX                ; acumula CX en AX.
printn                       ; salto de linea

; imprime el resultado de la suma
print '          La suma es:'
CALL   print_num             ; imprime varlor de AX.
RET                          ; vuelve al sistema operativo.

;--- definicion de procedimientos de impresion
;--- y captura de datos de libreria emu8086.inc
```

```
DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS          ; definir si se usa print_num.
DEFINE_PTHIS

END                           ; directiva para detener el compilador.
```

2.- Realice un programa que pida por pantalla dos números, los sume, y presente el resultado. Deberá hacer uso de la librería emu8086.inc y hacer llamadas a las funciones:

```
; Ejercicio Calculadora simple. (modificacion de ejemplo incluido emu8086)
; incluye opcion para salir y ejecuta iterativamente la calculadora.
; Se incluyen funciones de emu8086.inc para no incluir toda la librería.
name "calc2"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; this maro is copied from emu8086.inc ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; this macro prints a char in AL and advances
; the current cursor position:
PUTC    MACRO   char
        PUSH    AX
        MOV     AL, char
        MOV     AH, 0Eh
        INT     10h
        POP     AX
ENDM
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

org 100h

jmp inicio

; define variables:

msg0 db 0Dh,0Ah, "note: calculator works with integer values only.",0Dh,0Ah, '$'
msg1 db 0Dh,0Ah, 0Dh,0Ah, 'Ingrese primer numero: $'
msg2 db "Ingrese el operador (q para salir):    +  -  *  /     : $"
msg3 db "Ingrese segundo numero: $"
msg4 db  0dh,0ah , 'El resultado del calculo es : $'
msg5 db  0dh,0ah ,'Gracias por usar nuestra calculadora! presione una tecla...
', 0Dh,0Ah, '$'
err1 db  "operador incorrecto!", 0Dh,0Ah , '$'
smth db  " and something.... $"

opr db '?'        ; operador puede ser: '+','-','*','/' o 'q' para salir.

; variables para numeros
num1 dw ?
num2 dw ?

inicio:
mov dx, offset msg0
mov ah, 9
int 21h
```

```
lea dx, msg2
mov ah, 09h     ; output string at ds:dx
int 21h

; leemos operador
mov ah, 1   ; single char input to AL.
int 21h
mov opr, al
cmp opr, 'q'      ; q - salir de programa
je exit

cmp opr, '*'
jb wrong_opr
cmp opr, '/'
ja wrong_opr

; new line:
putc 0Dh
putc 0Ah

lea dx, msg1
mov ah, 09h    ; output string at ds:dx
int 21h

call scan_num           ; obtener numero multidigito desde teclado y almacena en
cx
mov num1, cx            ; guardo numero 1

; new line:
putc 0Dh
putc 0Ah


lea dx, msg3            ; output of a string at ds:dx
mov ah, 09h
int 21h

call scan_num           ; obtener numero multidigito desde teclado y almacena en
cx
mov num2, cx            ; guardo numero 2

lea dx, msg4
mov ah, 09h      ; Salida por pantalla leyenda de resultado
int 21h

; calculos:

cmp opr, '+'
je suma

cmp opr, '-'
je resta

cmp opr, '*'
je multiplicacion

cmp opr, '/'
je division
```

```
; ninguno de anteriores, entonces ERROR
wrong_opr:
lea dx, err1
mov ah, 09h     ; output string at ds:dx
int 21h

exit:
; output of a string at ds:dx
lea dx, msg5
mov ah, 09h
int 21h

; wait for any key...
mov ah, 0
int 16h

MOV AX,0600H      ; Peticion para limpiar pantalla
MOV BH,A2H  ; Color de letra ==2 "Verde 8 Azul Claro"
MOV CX,0000H      ; Se posiciona el cursor en Ren=0 Col=0
MOV DX,184FH      ; Cursor al final de la pantalla Ren=24(18)

INT 10H ; INTERRUPCION AL BIOS

jmp inicio

ret  ; return back to os.

suma:
mov ax, num1
add ax, num2
call print_num    ; print ax value.
jmp exit

resta:
mov ax, num1
sub ax, num2
call print_num    ; print ax value.
jmp exit

multiplicacion:
mov ax, num1
imul num2 ; (dx ax) = ax * num2.
call print_num    ; print ax value.
; dx is ignored (calc works with tiny numbers only).
jmp exit
division:
; dx is ignored (calc works with tiny integer numbers only).
mov dx, 0
mov ax, num1
idiv num2  ; ax = (dx ax) / num2.
cmp dx, 0
jnz approx
call print_num    ; print ax value.
jmp exit
approx:
call print_num    ; print ax value.
lea dx, smth
mov ah, 09h    ; output string at ds:dx
int 21h
jmp exit
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; these functions are copied from emu8086.inc ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; gets the multi-digit SIGNED number from the keyboard,
; and stores the result in CX register:
SCAN_NUM        PROC    NEAR
        PUSH    DX
        PUSH    AX
        PUSH    SI

        MOV     CX, 0

        ; reset flag:
        MOV     CS:make_minus, 0

next_digit:

        ; get char from keyboard
        ; into AL:
        MOV     AH, 00h
        INT     16h
        ; and print it:
        MOV     AH, 0Eh
        INT     10h

        ; check for MINUS:
        CMP     AL, '-'
        JE      set_minus

        ; check for ENTER key:
        CMP     AL, 0Dh  ; carriage return?
        JNE     not_cr
        JMP     stop_input
not_cr:


        CMP     AL, 8                   ; 'BACKSPACE' pressed?
        JNE     backspace_checked
        MOV     DX, 0                   ; remove last digit by
        MOV     AX, CX                  ; division:
        DIV     CS:ten                  ; AX = DX:AX / 10 (DX-rem).
        MOV     CX, AX
        PUTC    ' '                     ; clear position.
        PUTC    8                       ; backspace again.
        JMP     next_digit
backspace_checked:


        ; allow only digits:
        CMP     AL, '0'
        JAE     ok_AE_0
        JMP     remove_not_digit
ok_AE_0:
        CMP     AL, '9'
        JBE     ok_digit
remove_not_digit:
        PUTC    8       ; backspace.
        PUTC    ' '     ; clear last entered not digit.
```

```
        PUTC    8       ; backspace again.
        JMP     next_digit ; wait for next input.
ok_digit:


        ; multiply CX by 10 (first time the result is zero)
        PUSH    AX
        MOV     AX, CX
        MUL     CS:ten                  ; DX:AX = AX*10
        MOV     CX, AX
        POP     AX

        ; check if the number is too big
        ; (result should be 16 bits)
        CMP     DX, 0
        JNE     too_big

        ; convert from ASCII code:
        SUB     AL, 30h

        ; add AL to CX:
        MOV     AH, 0
        MOV     DX, CX      ; backup, in case the result will be too big.
        ADD     CX, AX
        JC      too_big2    ; jump if the number is too big.

        JMP     next_digit

set_minus:
        MOV     CS:make_minus, 1
        JMP     next_digit

too_big2:
        MOV     CX, DX      ; restore the backuped value before add.
        MOV     DX, 0       ; DX was zero before backup!
too_big:
        MOV     AX, CX
        DIV     CS:ten  ; reverse last DX:AX = AX*10, make AX = DX:AX / 10
        MOV     CX, AX
        PUTC    8       ; backspace.
        PUTC    ' '     ; clear last entered digit.
        PUTC    8       ; backspace again.
        JMP     next_digit ; wait for Enter/Backspace.


stop_input:
        ; check flag:
        CMP     CS:make_minus, 0
        JE      not_minus
        NEG     CX
not_minus:

        POP     SI
        POP     AX
        POP     DX
        RET
make_minus      DB      ?       ; used as a flag.
SCAN_NUM        ENDP
```

```
; this procedure prints number in AX,
; used with PRINT_NUM_UNS to print signed numbers:
PRINT_NUM       PROC    NEAR
        PUSH    DX
        PUSH    AX

        CMP     AX, 0
        JNZ     not_zero

        PUTC    '0'
        JMP     printed

not_zero:
        ; the check SIGN of AX,
        ; make absolute if it's negative:
        CMP     AX, 0
        JNS     positive
        NEG     AX

        PUTC    '-'

positive:
        CALL    PRINT_NUM_UNS
printed:
        POP     AX
        POP     DX
        RET
PRINT_NUM       ENDP



; this procedure prints out an unsigned
; number in AX (not just a single digit)
; allowed values are from 0 to 65535 (FFFF)
PRINT_NUM_UNS   PROC    NEAR
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX

        ; flag to prevent printing zeros before number:
        MOV     CX, 1

        ; (result of "/ 10000" is always less or equal to 9).
        MOV     BX, 10000       ; 2710h - divider.

        ; AX is zero?
        CMP     AX, 0
        JZ      print_zero

begin_print:

        ; check divider (if zero go to end_print):
        CMP     BX,0
        JZ      end_print

        ; avoid printing zeros before number:
```

```
        CMP     CX, 0
        JE      calc
        ; if AX<BX then result of DIV will be zero:
        CMP     AX, BX
        JB      skip
calc:
        MOV     CX, 0   ; set flag.

        MOV     DX, 0
        DIV     BX      ; AX = DX:AX / BX   (DX=remainder).

        ; print last digit
        ; AH is always ZERO, so it's ignored
        ADD     AL, 30h    ; convert to ASCII code.
        PUTC    AL


        MOV     AX, DX  ; get remainder from last div.

skip:
        ; calculate BX=BX/10
        PUSH    AX
        MOV     DX, 0
        MOV     AX, BX
        DIV     CS:ten  ; AX = DX:AX / 10   (DX=remainder).
        MOV     BX, AX
        POP     AX

        JMP     begin_print

print_zero:
        PUTC    '0'

end_print:

        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
PRINT_NUM_UNS   ENDP



ten             DW      10      ; used as multiplier/divider by SCAN_NUM &
PRINT_NUM_UNS.




GET_STRING      PROC    NEAR
PUSH    AX
PUSH    CX
PUSH    DI
PUSH    DX

MOV     CX, 0                   ; char counter.
```

```
CMP     DX, 1                   ; buffer too small?
JBE     empty_buffer            ;

DEC     DX                      ; reserve space for last zero.


;============================
; Eternal loop to get
; and processes key presses:

wait_for_key:

MOV     AH, 0                   ; get pressed key.
INT     16h

CMP     AL, 0Dh                  ; 'RETURN' pressed?
JZ      exit_GET_STRING


CMP     AL, 8                   ; 'BACKSPACE' pressed?
JNE     add_to_buffer
JCXZ    wait_for_key            ; nothing to remove!
DEC     CX
DEC     DI
PUTC    8                       ; backspace.
PUTC    ' '                     ; clear position.
PUTC    8                       ; backspace again.
JMP     wait_for_key

add_to_buffer:

        CMP     CX, DX          ; buffer is full?
        JAE     wait_for_key    ; if so wait for 'BACKSPACE' or 'RETURN'...

        MOV     [DI], AL
        INC     DI
        INC     CX

        ; print the key:
        MOV     AH, 0Eh
        INT     10h

JMP     wait_for_key
;============================

exit_GET_STRING:

; terminate by null:
MOV     [DI], 0

empty_buffer:

POP     DX
POP     DI
POP     CX
POP     AX
RET
GET_STRING      ENDP
```