

11 de octubre de 2020

# Manual Técnico

## CALCULADORA

Elaborado por: Mariana Sic 201504051

### Encabezado

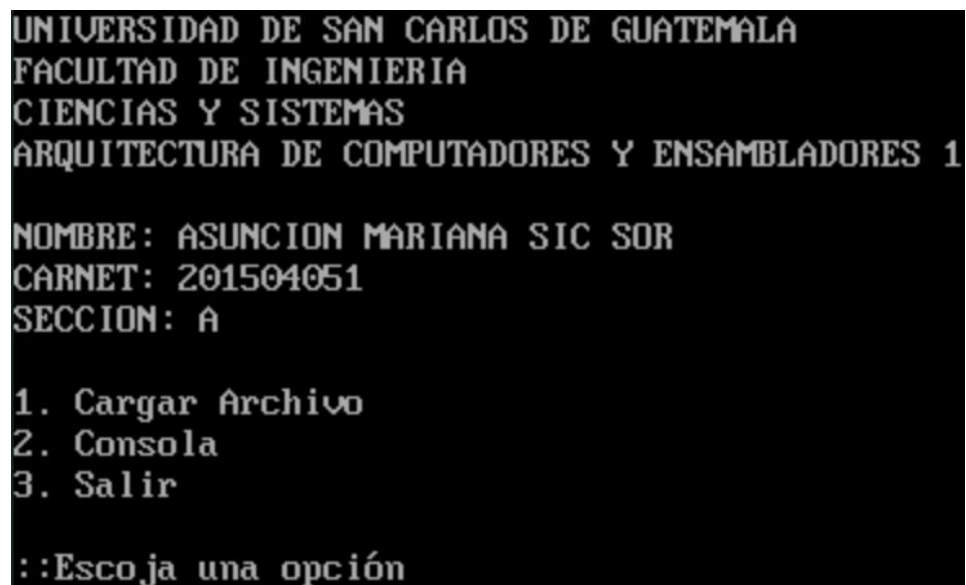
Para el encabezado se utilizaron tres variables: “encab”, “enc”, “encab2”. Las cuales tienen el contenido solicitado para el encabezado

```
encab db 10,10,13,'UNIVERSIDAD DE SAN CARLOS DE GUATEMALA',10,13,'FACULTAD  
DE INGENIERIA',10,13,'CIENCIAS Y SISTEMAS',10,13,'ARQUITECTURA DE  
COMPUTADORES Y ENSAMBLADORES 1','$'
```

```
enc db 10,10,13,'NOMBRE: ASUNCION MARIANA SIC SOR',10,13,'CARNET:  
201504051',10,13,'SECCION: A','$'
```

```
encab2 db 10,13,10,13,'1. Cargar Archivo',10,13,'2. Consola',10,13,'3.  
Salir',10,13,10,13,'::Escoja una opci',162,'n ','$'
```

Dando como resultado:



```
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA  
FACULTAD DE INGENIERIA  
CIENCIAS Y SISTEMAS  
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1  
  
NOMBRE: ASUNCION MARIANA SIC SOR  
CARNET: 201504051  
SECCION: A  
  
1. Cargar Archivo  
2. Consola  
3. Salir  
  
::Escoja una opción
```

Donde se debe de ingresar la opción deseada, estas opciones se detallan en las siguientes secciones:

1. Opción 1: Carga de un archivo JSON. [Ver Detalle](#)
2. Opción 2: Consola. [Ver Detalle](#)
3. Opción 3: Salir. [Ver Detalle](#)

## Carga de un archivo JSON

Para la carga de un archivo JSON, primero se debió haber presionado la opción 1

```
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
CIENCIAS Y SISTEMAS
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1
```

```
NOMBRE: ASUNCION MARIANA SIC SOR
CARNET: 201504051
SECCION: A
```

```
1. Cargar Archivo
2. Consola
3. Salir
```

```
::Escoja una opción 1
```

```
----- CARGAR ARCHIVO -----
```

```
::Ingresa nombre archivo a cargar
```

El archivo a cargar se llama P.JSON y contiene lo siguiente:

```
{
  "padre":
  [
    {
      "operacion1":{
        "MUL":{
          "+":{
            "#":4,
            "-":{
              "#":34,
              "div":{
                "#":-30,
                "#":2
              }
            }
          }
        }
      }
    }
  ]
}
```

```

    }
    },
    "#":4
  }
},
{
  "operacion2":{
    "aDD":{
      "/":{
        "sUb":{
          "#":30,
          "#":5
        },
        "#":-4
      },
      "id":"operacion1"
    }
  },
  "operacion3":{
    "-":{
      "#":-40,
      "#":-10
    }
  },
  "operacion4":{
    "MUL":{
      "id":"operacion3",
      "dIv":{
        "id":"operacion2",
        "#":2
      }
    }
  }
}
]
}

```

Se ingresa el nombre de la ruta y pronto estará cargado

```

----- CARGAR ARCHIVO -----
::Ingresa nombre archivo a cargar P.JSO
!!! Archivo cargado con éxito !!!

```

Para la carga del archivo se utilizó el macro llamado calcularJson:

```
calcularJson macro archivo
    local LEER, STRING, FIN, QUIZANUM, FIJONUM, NOESNUM
    ;resultados es el arr res
    xor si,si

    LEER:
        cmp si,di
        jge FIN
        cmp archivo[si],'"'
        je STRING
        cmp archivo[si],'- '
        je NEGATIVO
        cmp archivo[si],48
        jge QUIZANUM
        inc si
        jmp LEER

    NEGATIVO:
        cleanArr arregloAux
        push di
        xor di,di
        mov arregloAux[di],'- '
        inc di
        inc si
        guardarNumero archivo
        escribirF handle2,sizeof arregloAux, arregloAux
        jmp LEER

    QUIZANUM:
        cmp archivo[si],57
        jle FIJONUM
        jmp NOESNUM

    FIJONUM:
        cleanArr arregloAux
        push di
        xor di,di
        guardarNumero archivo
        escribirF handle2,sizeof arregloAux, arregloAux
        jmp LEER

    NOESNUM:
        inc si
        jmp LEER

    STRING:
        inc si
```

```

        cleanArr arregloAux
        obtenerEntreComilla archivo
        escribirF handle2,sizeof arregloAux, arregloAux
        jmp LEER

FIN:
        ;fin de la lectura del archivo
endm

```

Junto con dos macros auxiliares, el primero se llama “obtenerEntreComilla”, el cual se encarga de obtener todo lo que esté entre comillas en el archivo JSON

```

obtenerEntreComilla macro archivo
    local INICIO, FIN, FIJOMAYUS, NOESMAYUS, QUIZAMAYUS

    push di
    push ax
    xor ax,ax
    xor di,di
    mov arregloAux[0],32
    inc di

    INICIO:
        cmp archivo[si],'"'
        je FIN
        cmp archivo[si],65
        jge QUIZAMAYUS
        mov al,archivo[si]
        mov arregloAux[di],al
        inc si
        inc di
        jmp INICIO

    QUIZAMAYUS:
        cmp archivo[si],90
        jle FIJOMAYUS
        jmp NOESMAYUS

    FIJOMAYUS:
        mov al,archivo[si]
        add al,32 ;para que sea minuscula
        mov arregloAux[di],al
        inc di
        inc si
        jmp INICIO

    NOESMAYUS:
        mov al,archivo[si]
        mov arregloAux[di],al
        inc di

```

```

        inc si
        jmp INICIO

FIN:
        inc si
        pop ax
        pop di
endm

```

Y por último, el macro “guardarNumero” el cual se encarga de almacenar los números dentro del archivo JSON

```

guardarNumero macro archivo
    local INICIO, QUIZANUM, FIJONUM, FIN
    push ax
    ;xor ax,ax
    INICIO:
        cmp archivo[si],48
        jge QUIZANUM
        jmp FIN

    QUIZANUM:
        cmp archivo[si],57
        jle FIJONUM
        jmp FIN

    FIJONUM:
        mov al,archivo[si]
        mov arregloAux[di],al
        inc si
        inc di
        jmp INICIO

    FIN:
        pop ax
        pop di
endm

```

Después de obtenidos los números e identificadores, se procede a darle la vuelta a todo el archivo para que se pueda leer de abajo para arriba, esto con el fin de realizar una operación postfija.

Para esto, se utiliza un macro llamado “analisis2” el cual se encarga de realizar el trabajo anteriormente descrito.

```

analisis2 macro
    local INICIO, FIN, ESID, NUMERALITO, MULTI, DIVIS, SUMA, RESTA

    xor di,di

    INICIO:
        cmp di,si
        jge FIN

        leerF sizeof arregloAux, arregloAux, handle2

        comparar arregloAux,numeral
        cmp ah,1b
        je NUMERALITO

        comparar arregloAux,idRes
        cmp ah,1b
        je ESID

        comparar arregloAux,addRes1
        cmp ah,1b
        je SUMA

        comparar arregloAux,subRes1
        cmp ah,1b
        je RESTA

        comparar arregloAux,mulRes1
        cmp ah,1b
        je MULTI

        comparar arregloAux,divRes1
        cmp ah,1b
        je DIVIS

        escribirF handleFichero,sizeof arregloAux,arregloAux
        inc di
        jmp INICIO

    SUMA:
        cleanArr arregloAux
        mov arregloAux[0],32
        mov arregloAux[1], '+'
        escribirF handleFichero, sizeof arregloAux, arregloAux
        inc di
        jmp INICIO

    RESTA:
        cleanArr arregloAux

```

```

    mov arregloAux[0], 32
    mov arregloAux[1], '-'
    escribirF handleFichero, sizeof arregloAux, arregloAux
    inc di
    jmp INICIO

MULTI:
    cleanArr arregloAux
    mov arregloAux[0], 32
    mov arregloAux[1], '*'
    escribirF handleFichero, sizeof arregloAux, arregloAux
    inc di
    jmp INICIO

DIVIS:
    cleanArr arregloAux
    mov arregloAux[0], 32
    mov arregloAux[1], '/'
    escribirF handleFichero, sizeof arregloAux, arregloAux
    inc di
    jmp INICIO

NUMERALITO:
    inc di
    jmp INICIO

ESID:
    leerF sizeof arregloAux, arregloAux, handle2
    inc di
    mov arregloAux[0], '!'
    escribirF handleFichero, sizeof arregloAux, arregloAux
    inc di
    jmp INICIO

FIN:
    mov dx, di
endm

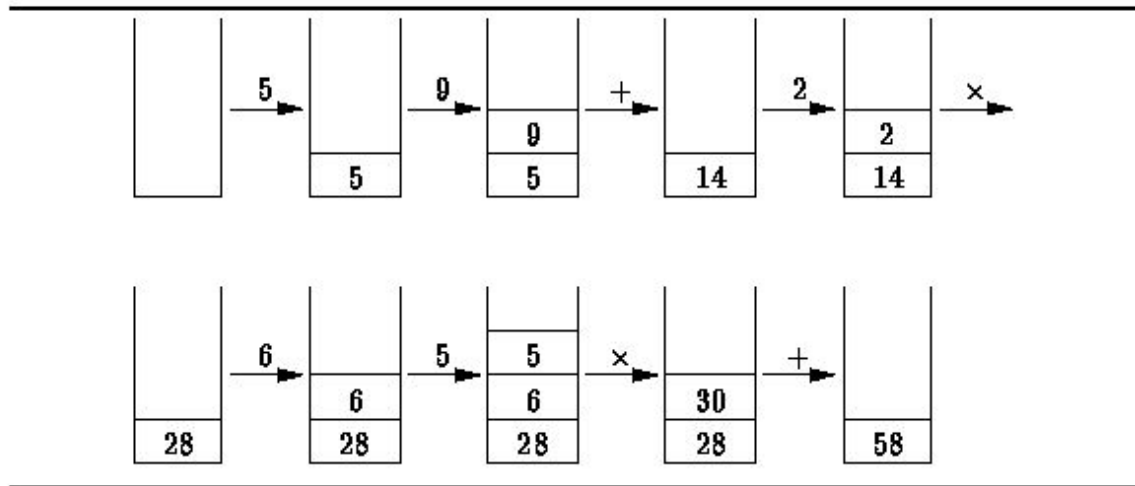
```

Finalmente, para operar se siguió el algoritmo de resolución expresiones en notación postfija haciendo uso de la pila:

1. Si el término es un valor, introducirlo a la pila
2. Si el término es un operador:
  - a. Sacar dos operadores de la pila
  - b. Aplicar el operador que corresponde
  - c. Meter el resultado a la pila



Este procedimiento lo ilustra la siguiente imagen



Y para realizar dicho algoritmo, se realizó en una macro llamada “operacionFinal”

```
operacionFinal macro
    local INICIO, FIN, ESCR, BUSCARID, IN2, FIN2, IN3, FIN3, SUMARS,
MULTIPLICARS, DIVIDIRS, RESTARS, SACARRESPUESTA, FINALITO

    cleanArr arregloAux
    abrirF rutaAux,handle
    leerF sizeof bufferLectura,bufferLectura,handle
    xor dx,dx
    mov bl,72
    div bl
    mov dx,ax
    cerrarF handle
    mov handle,00h

    abrirF rutaAux,handle
    mov bx,0

    mov variable,dx

    INICIO:
        ;cleanArr arregloAux
        cmp bx,variable
        jge FIN
        ;darle la vuelta para operar
        leerF sizeof arregloAux, arregloAux, handle

        cmp arregloAux[1], '+'
```

```

je ESCR
cmp arregloAux[1], '-'
je ESCR
cmp arregloAux[1], '*'
je ESCR
cmp arregloAux[1], '/'
je ESCR
cmp arregloAux[0], '!'
je BUSCARID

ConvertirAscii arregloAux
mov dx, ax
push dx

inc bx
jmp INICIO

ESCR:
xor ax, ax
mov al, arregloAux[1]
mov dx, 1000
add dx, ax
push dx
inc bx
jmp INICIO

BUSCARID:
xor dx, dx
mov dl, arregloAux[0]
ConvertirSumaAscii arregloAux ;en ax esta la suma de asciis del id,
hay que restarle 1
sub ax, 1
searchID
push dx
inc bx
jmp INICIO

FIN:
cerrarF handle
borrarF rutaAux

mov handle, 00h
crearF rutaAux, handle
abrirF rutaAux, handle
xor bx, bx

```

```

////////////////////////////////////
//
IN2:
    cmp bx,variable
    jge FIN2

    pop dx
    mov ax,dx
    ConvertirString resultadosTmp
    escribirF handle, sizeof resultadosTmp, resultadosTmp
    inc bx
    jmp IN2

FIN2:
    cerrarF handle
    mov handle,00h

    abrirF rutaAux,handle
    xor bx,bx
    mov auxWord[0], 'N'

IN3:
    cmp bx,variable
    jge FIN3

    leerF sizeof arregloAux, arregloAux, handle

    ConvertirAscii arregloAux
    cmp ax,1042 ;multiplicacion
    je MULTIPLICARS
    cmp ax,1043 ;suma
    je SUMARS
    cmp ax,1045 ;resta
    je RESTARS
    cmp ax,1047 ;division
    je DIVIDIRS

    mov dx,ax
    push dx

    inc bx
    jmp IN3

MULTIPLICARS:
    pop dx
    mov ax,dx
    pop dx

```

```

mov cx,dx
imul cx
mov dx,ax
push dx
mov auxWord[0], 'Y'
inc bx
jmp IN3

```

SUMARS:

```

pop dx
mov ax,dx
pop dx
add ax,dx
mov dx,ax
push dx
mov auxWord[0], 'Y'
inc bx
jmp IN3

```

RESTARTS:

```

pop dx
mov ax,dx
pop dx
sub ax,dx
mov dx,ax
push dx
mov auxWord[0], 'Y'
inc bx
jmp IN3

```

DIVIDIRS:

```

pop dx
mov ax,dx
pop dx
mov cx,dx
xor dx,dx
idiv cx
mov dx,ax
push dx
mov auxWord[0], 'Y'
inc bx
jmp IN3

```

FIN3:

```

cerrarF handle
mov handle, 00h
borrarF rutaAux
cmp auxWord[0], 'Y'
je SACARRESPUESTA

```

```

        jmp FINALITO

SACARRESPUESTA:
    pop dx ;resultado final
    mov ax,dx
    colocarRespuesta resultadosFinales
    ;ConvertirString resultadosTmp
    ;print resultadosTmp
    ;getChar

FINALITO:
    ;getChar

endm

```

Y los resultados se almacenan en dos arreglos de tipo “palabra” (16 bits) de la siguiente manera:

Supongamos que el identificador es “operacion1” primero se procede hacer la sumatoria de código ASCII de la palabra. Para esto se utiliza la siguiente macro:

```

ConvertirSumaAscii macro numero
    LOCAL INICIO,FIN

    push si
    xor ax,ax

    xor si,si
    INICIO:
        xor cx,cx
        mov cl,numero[si]
        cmp cl,'$'
        je FIN

        add ax,cx
        inc si
        jmp INICIO

    FIN:
        pop si
endm

```

El resultado de la sumatoria está en el registro AX

Todo el procedimiento anterior, se resume de la siguiente manera:

SÍMBOLO	o	p	e	r	a	c	i	o	n	1
CODIGO ASCII	111	112	101	114	97	99	105	111	110	49

Haciendo la suma de los códigos anteriores, el resultado es de 1,009. Por lo tanto, se almacena en cierta posición del arreglo de identificadores.

POSICIÓN	0	1	2	3	4	5	6	7	8	9
SUMA ASCII	964	789	1009	3289	154	0	0	0	0	0

De esta manera queda el arreglo de identificadores si se siguieran agregando más operaciones, los espacios con **0** representan disponibilidad, es decir, están vacíos.

Esto se realiza con el fin de que en el arreglo de respuestas, el identificador y la respuesta queden en la misma posición. Supongamos que el resultado de “operacion1” es -568, este resultado se debe de almacenar en la posición 2 pero del arreglo de respuestas finales.

Para que coincidan, se utiliza una macro que busca el primer espacio disponible en el arreglo (es decir, la posición donde el contenido sea **0**):

**colocarRespuesta** macro arreglo

local INICIO, FIN

push si  
push bx  
xor si,si

INICIO:  
mov bx,arreglo[si]  
cmp bx,00h  
je FIN

inc si  
inc si  
jmp INICIO

FIN:  
mov arreglo[si],ax

pop bx  
pop si

endm

## Consola

Para escribir comandos que corresponden a la consola, se debe de escribir la opción 2 del menú principal, seguido se entra a la consola:

```
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
CIENCIAS Y SISTEMAS
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1

NOMBRE: ASUNCION MARIANA SIC SOR
CARNET: 201504051
SECCION: A

1. Cargar Archivo
2. Consola
3. Salir

::Escoja una opción 2

----- CONSOLA -----

» _
```

La etiqueta que hace lo anterior posible, es la siguiente

```
CONSOLA:
    print msjOpc2
    print console
    cleanArr arregloAux
    ObtenerTexto arregloAux

    comparar arregloAux,exit
    cmp ah,1b
    je EXITCONSOLA

    comparar arregloAux,sh
    cmp ah,1b
    je SHOWCMD

    jne ErrorCmd
    jmp CONSOLA
```

Los comandos admitidos en consola son los siguientes:

1. EXIT: Si se escribe este comando, se sale de la consola y regresa al menú principal

Se realiza mediante la siguiente etiqueta:

```
EXITCONSOLA:  
    jmp MenuPrincipal
```

Y como resultado en el programa:

```
1. Cargar Archivo  
2. Consola  
3. Salir  
::Escoja una opción 2  
  
----- CONSOLA -----  
  
» EXIT  
  
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA  
FACULTAD DE INGENIERIA  
CIENCIAS Y SISTEMAS  
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1  
  
NOMBRE: ASUNCION MARIANA SIC SOR  
CARNET: 201504051  
SECCION: A  
  
1. Cargar Archivo  
2. Consola  
3. Salir  
::Escoja una opción _
```

2. SHOW ID: En este comando, la palabra **ID** representa algún identificador, este identificador puede ser de alguna operación hija o de la operación padre.
  - a. Si se coloca la operación hija, por ejemplo, "operacion4":

```
----- CONSOLA -----  
  
» SHOW operacion4  
  
El resultado de operacion4 es: -3090  
  
----- CONSOLA -----  
  
» _
```



Esto lo que hacer es obtener el identificador que se ingresó, utilizar el macro ConvertirSumaAscii y verificar que exista en el arreglo de identificadores para luego ir a esa posición y obtener el resultado del arreglo de resultados. Esto se hace mediante el macro:

```
searchID macro
    ;lo que devuelve el id, se coloca en dx
    local INICIO, NOENCONTRADO, ENCONTRADO, FIN
    push si
    push bx
    xor si,si

    INICIO:
        mov bx,idsFinales[si]
        cmp bx,00h
        je NOENCONTRADO
        cmp bx,ax
        je ENCONTRADO

        inc si
        inc si
        jmp INICIO

    NOENCONTRADO:
        mov dx,0
        jmp FIN

    ENCONTRADO:
        mov dx,resultadosFinales[si]

    FIN:

    pop bx
    pop si
endm
```

El resultado, se encuentra en el registro DX. Si no existe, se mostrará un mensaje de error

```
print elresid
print arregloAux
print esRes
ConvertirSumaAscii arregloAux
add ax,32
searchID

cmp dx,0
je NOEXISTE

mov ax,dx
ConvertirString resultadosTmp
print resultadosTmp
```

```

    jmp CONSOLA

NOEXISTE:
    print noexisteMsj
    jmp CONSOLA

```

En dado caso la operacion no exista:

```

----- CONSOLA -----

» SHOW operacion5

El resultado de operacion5 es: La operación que ingresaste no existe

----- CONSOLA -----

» _

```

b. Si se coloca el nombre de la operacion padre, por ejemplo, “padre”:

```

----- CONSOLA -----

» SHOW padre

!! Reporte Generado !!

----- CONSOLA -----

»

```

Se genera un reporte en un archivo llamado “REPORTE.JSON” en formato JSON, el cual contiene lo siguiente:

```

{
  "reporte": {
    "alumno": {
      "nombre": "Asuncion Mariana Sic Sor",
      "carnet": 201504051,
      "seccion": "A",
      "curso": "Arquitectura de Computadores y Ensambladores 1"
    },
    "fecha": {
      "dia": 11,
      "mes": 10,
      "a~o": 2020
    },
    "hora": {
      "hora": 19,
      "minutos": 47,

```

```

        "segundos":12
    },
    "resultados":{
        "media":675,
        "mediana":0,
        "moda":0,
        "menor":-3090,
        "mayor":212
    },
    "padre":[
        {
            "operacion1":212
        },
        {
            "operacion2":206
        },
        {
            "operacion3":-30
        },
        {
            "operacion4":-3090
        }
    ]
}

```

Este se genera con la ayuda del siguiente macro

```

generarReporte macro
    mov handle,00h
    crearF answers,handle
    escribirF handle, sizeof rep1, rep1
    escribirF handle, sizeof rep2, rep2
    escribirFecha

    getMedia
    escribirA resultadosTmp, handle

    escribirF handle, sizeof medRep, medRep
    getMediana
    escribirA resultadosTmp, handle

    escribirF handle, sizeof modRep, modRep
    getModa
    escribirA resultadosTmp, handle

    escribirF handle, sizeof menRep, menRep
    getMenor
    escribirA resultadosTmp, handle

```

```

    escribirF handle, sizeof mayRep, mayRep
    getMayor
    escribirA resultadosTmp, handle

    escribirF handle, sizeof op1, op1
    escribirA nombrePadre, handle
    escribirF handle, sizeof op2, op2

    getOperaciones

    escribirF handle, sizeof finRep, finRep
    cerrarF handle
    mov handle, 00h
endm

```

Y para obtener el arreglo de los resultados de las operaciones en el archivo, se utiliza la siguiente macro

```

getOperaciones macro
    local INICIO, FIN, ESCRIBCOMA

    xor si, si
    xor di, di

    INICIO:
        mov ax, resultadosFinales[si]
        cmp ax, 00h
        je FIN

        ConvertirString resultadosTmp

        cmp si, 0
        jg ESCRIBCOMA

    NORMAL:
        escribirF handle, sizeof inOperaciones, inOperaciones
        ; agarrar id de las operaciones
        escribirF handle, sizeof middleOperaciones, middleOperaciones
        escribirA resultadosTmp, handle
        inc si
        inc si
        jmp INICIO

    ESCRIBCOMA:
        escribirF handle, sizeof finOperaciones, finOperaciones
        escribirF handle, sizeof coma, coma
        jmp NORMAL

```

```

FIN:
    escribirF handle, sizeof finOperaciones, finOperaciones

```

endm

3. SHOW MEDIA: Con este comando se obtiene la media aritmética de los resultados obtenidos, con la siguiente fórmula:

$$\bar{x} = \frac{\sum x_i}{n}$$

```

----- CONSOLA -----
» SHOW MEDIA
::La media es: 675
----- CONSOLA -----
» _

```

Para esto, se utiliza la siguiente macro:

```

getMedia macro
    local INICIO, FIN, NEGATIVO, FINSI

    xor si,si
    xor ax,ax
    xor cx,cx

    INICIO:
        mov bx,resultadosFinales[si]
        cmp bx,00h
        je FIN

        add ax,bx

        inc si
        inc si
        inc cx
        jmp INICIO

    FIN:
        cmp ax,0
        jl NEGATIVO

```

```

        jmp FINSI

NEGATIVO:
        neg ax

FINSI:
        xor dx,dx
        idiv cx
        ConvertirString resultadosTmp
endm

```

4. SHOW MODA: muestra la moda de los resultados obtenidos



```

----- CONSOLA -----
» SHOW MODA
::La moda es:
----- CONSOLA -----
»

```

Para esto, se utiliza la macro

```

getModa macro
    local INICIO, FIN, INICIO2, FIN2, ESMENOR

    xor si,si
    xor ax,ax
    xor cx,cx

    INICIO:
        mov ax,0

    FIN:
        ConvertirString resultadosTmp
endm

```

5. SHOW MEDIANA: Se obtiene la mediana de los resultados obtenidos

```
----- CONSOLA -----
» SHOW MEDIANA
::La mediana es:
----- CONSOLA -----
» _
```

Esto con ayuda de la macro

```
getMediana macro
    local INICIO, FIN, INICIO2, FIN2, ESMENOR

    xor si,si
    xor ax,ax
    xor cx,cx

    INICIO:
        mov ax,0

    FIN:
        ConvertirString resultadosTmp
endm
```

6. SHOW MAYOR: Muestra el número mayor respecto a los números de los resultados obtenidos. Con ayuda de la macro

```
getMayor macro
    local INICIO, FIN, ESMAYOR

    xor si,si
    xor ax,ax
    xor cx,cx

    INICIO:
        mov bx,resultadosFinales[si]
        cmp bx,00h
        je FIN

        cmp bx,ax
        jg ESMAYOR

        inc si
        inc si
        jmp INICIO
```

```

ESMAYOR:
    mov ax,bx
    inc si
    inc si
    jmp INICIO

FIN:
    ConvertirString resultadosTmp

```

endm

```

----- CONSOLA -----
» SHOW MAYOR
::El número mayor es: 212
----- CONSOLA -----
»

```

7. SHOW MENOR: Muestra el número mayor respecto a los números de los resultados obtenidos. Con ayuda de la macro

```

getMenor macro
    local INICIO, FIN, ESMAYOR

    xor si,si
    xor ax,ax
    xor cx,cx

    INICIO:
        mov bx,resultadosFinales[si]
        cmp bx,00h
        je FIN

        cmp bx,ax
        jl ESMAYOR

        inc si
        inc si
        jmp INICIO

    ESMAYOR:
        mov ax,bx
        inc si
        inc si

```



```

        jmp INICIO

FIN:
    ConvertirString resultadosTmp

endm

```

```

----- CONSOLA -----

» SHOW MENOR

::El número menor es: -3090

----- CONSOLA -----

»

```

## Salir

Para realizar la acción de salir del programa en general, se utilizó la función 4CH de la interrupción 21H junto con la etiqueta "SALIR" y un mensaje "msjOpc3" que se despide del programa

```

SALIR:
    print msjOpc3
    mov ah,4ch
    int 21h

```

Funciona de la siguiente manera:

```

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
CIENCIAS Y SISTEMAS
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1

NOMBRE: ASUNCION MARIANA SIC SOR
CARNET: 201504051
SECCION: A

1. Cargar Archivo
2. Consola
3. Salir

::Escoja una opción 3

----- ADIOS -----

```