



Manual Técnico

24 MARZO

José Ottoniel Sincal Guitzol
201801375



INDICE

Objetivo	2
Descripción.....	2
Entorno de Trabajo.....	2
Descripción de Macros.....	3
getChar	3
obtenerTexto	3
obtenerNumero	4
printNumero	5
obtenerFactorialAux	6
analizar	7
comparacion	8
obtenerStringNumero.....	9
Descripción Procedimientos	10
Etiqueta encabezado	10
Etiqueta menú	10
Etiqueta cargarArchivo	11
Etiqueta cerrarArchivo	12
Etiqueta analizarArchivo	12
Etiqueta esNumero	13
Etiqueta resultadoArchivo	13
Etiqueta calculadora	14
Etiqueta sumar, restar, multiplicar y dividir	14
Etiqueta pedirMas	15
Etiqueta mostrarResultado	16
Etiqueta factorial	16

Objetivo

Otorgar soporte y facilidad de entendimiento a toda persona la cual posee conocimientos informáticos y que no se le dificulte el poder actualizar o estudiar a detalle el correcto funcionamiento de la aplicación.

Descripción

La aplicación consiste en una sencilla en consola utilizando programación a bajo nivel, para la presente práctica será una calculadora en la cual se manejarán los signos en las operaciones aritméticas, suma (+), resta (-), multiplicación (*) y división (/).

Entorno de Trabajo

Dicha práctica se realizó utilizando programación a bajo nivel, se hizo uso del ensamblador MASM junto con el emulador DOSBox para poder correr el programa creado.

Descripción de Macros

getChar

Captura un carácter el cual se ingresa por medio del teclado.

- Se hace el llamado a la interrupción 01h el cual espera la leída de un carácter de entrada.

```
1  getChar macro                ; obtiene el caracter
2      mov ah, 01h              ; se guarda en al en codigo hex
3      int 21h
4  endm
```

obtenerTexto

Captura el texto que estemos ingresando actualmente en pantalla, hasta que reconozca un salto de línea.

- El registro si, se utilizó para poder guardar cada carácter dentro de un arreglo, en una posición determinada.
- Se manda a llamar al macro getChar para obtener un carácter.
- Compara si el carácter es un salto de línea, si lo es termina el proceso y añade el carácter de '\$' para finalizar la cadena de entrada. De lo contrario se guarda el carácter dentro del arreglo en la posición *si* luego se incrementa el contador y repite el proceso.

```
1  obtenerTexto macro buffer
2      LOCAL ObtenerChar, endTexto
3      xor si, si                ; xor si, si = mov si, 0
4
5      ObtenerChar:
6          getChar
7          cmp al, 0dh            ; ascii de salto de linea hex
8          je endTexto
9          mov buffer[si], al     ; mov destino, fuente
10         inc si                 ; si = si + 1
11         jmp ObtenerChar
12
13     endTexto:
14         mov al, 24h            ; ascii del signo dolar $
15         mov buffer[si], al
16     endm
```

obtenerNumero

Lee el número que se ingresa en pantalla en un rango de [-99, 99].

- Hace uso del macro getChar para pedir caracteres, evalúa si es primero es un signo menos, lo cual indica que es un número negativo, de lo contrario salta a la etiqueta de positivo.
- Si es positivo pide el segundo dígito.
- Si es negativo vuelve a pedir ambos. Para posteriormente negarlos.
- A la hora de convertir se le resta 30h (valor en hexadecimal = 48 en decimal) a los 2 dígitos para obtener su verdadero valor, ya que se reconoce el código ASCII como entrada, luego se multiplica por 10 al primero y posteriormente se suma el valor del segundo.

```
1  obtenerNumero macro numero, diez          ;etiqueta a la que regresa el salto
2      LOCAL isPositive, isNegative, NegateNumber, getSign
3
4      getSign:
5          getChar
6          cmp al, '-'
7          je isNegative
8
9      isPositive:
10         sub al, 30h
11         mov bl, 10
12         mul diez
13         mov numero, al ; ya se tiene el primer digito ahora el segundo
14         mov ah, 01h
15         int 21h
16         sub al, 30h
17         add numero, al
18         neg numero
19         jmp NegateNumber
20
21     isNegative:
22         mov ah, 01h
23         int 21h
24         sub al, 30h
25         mul diez
26         mov numero, al ; ya se tiene el primer digito ahora el segundo
27         mov ah, 01h
28         int 21h
29         sub al, 30h
30         add numero, al
31
32     NegateNumber:
33         neg numero
34
35     endm
```

printNumero

Imprime un número de dos dígitos en pantalla.

- Para ello se debe imprimir dígito por dígito.
- Primero se evalúa el número para saber si es positivo o negativo.
- Si es negativo se imprime antes el signo '-' (2dh en hexadecimal) y luego se niega el número para después imprimirlo.
- Luego de negar el numero o de saber que es positivo se imprime el número, para ello se divide dentro de 10 para obtener el primer dígito y guardar el residuo, a ambos dígitos se le suma el valor de 30h (valor en hexadecimal = 48 decimal) para obtener su valor real en código ASCII e imprimir el número con la interrupción 02h.

```
1  printNum macro numero, residuo, diez
2      LOCAL evaluateNumber, printSign, printNumero
3
4      evaluateNumber:
5          mov bl, numero
6          test bl, bl
7          jns printNumero
8
9      printSign:
10         mov ah, 02h
11         mov dx, 2dh
12         int 21h
13         neg numero
14
15     printNumero:
16         mov al, numero                ; el valor a convertir debe estar AX o AL, depende del tipo
17         cbw                          ; haciendo la conversion de Byte a Word
18         div diez                      ; resultado en AL y residuo en AH
19         mov residuo, ah
20         mov dl, al
21         add dl, 30h                  ; añadiendo lo substraído
22         mov ah, 02h
23         int 21h
24         mov dl, residuo
25         add dl, 30h                  ; añadiendo lo substraído
26         mov ah, 02h
27         int 21h
28     endm
```

obtenerFactorialAux

Obtiene el factorial de un número especificado.

- Primero verifica si el número es menor o igual a 1. Si lo es retorna un 1 como resultado.
- De lo contrario entramos a un ciclo el cual se repetirá hasta que el registro cx sea mayor al número indicado.
- Dentro del ciclo se multiplica el valor hasta que el valor a multiplicar sea 1, que es cuando se sale del ciclo.

```
1  obtenerFactorialAux macro resf
2      LOCAL ciclo, endCiclo, return1
3
4      mov al, resf
5
6      cmp al, 1
7      jbe return1
8
9      mov ah, 0
10     mov bx, ax
11
12     ciclo:
13         dec bx
14         mul bx
15         cmp bx, 1
16         jne ciclo
17         mov cx, ax
18         jmp endCiclo
19
20     return1:
21         mov cl, 1
22
23     endCiclo:
24         mov resf, cl
25
26     endm
```

analizar

Analiza cada una de las etiquetas que vienen dentro del archivo con extensión .xml.

- El parámetro arreglo es la cadena de entrada o lo contenido dentro del archivo, y arregloAux es donde se concatenarán las etiquetas.
- Primero evalúa si el carácter en la posición si es igual a '<' para empezar a concatenar, de lo contrario sigue recorriendo el arreglo hasta encontrar dicho carácter.
- Ya encontrado el carácter evaluar si el siguiente es '>', sino lo es concatena y sigue recorriendo para verificar si sigue concatenando o no.
- Al encontrar el carácter de cierre se sale y por ultimo se agrega el carácter de fin de cadena '\$'.

```
1  analizar macro arreglo, arregloAux
2      local evaluar, concatenar, salida
3      evaluar:
4          mov si, contG
5          mov di, 0
6
7          mov al, arreglo[si]
8          cmp al, '<'
9          je concatenar
10         inc si
11         inc contG
12         jmp evaluar
13
14     concatenar:
15         inc si
16         inc contG
17
18         mov bl, arreglo[si]
19         cmp bl, '>'
20         je salida
21
22         mov arregloAux[di], bl
23         inc di
24
25         jmp concatenar
26
27     salida:
28         inc si
29         inc contG
30         mov al, 24h      ; ascii del signo dolar $
31         mov arregloAux[di], al
32     endm
```


comparacion

Compara la etiqueta obtenida en el método de analizar.

- Comprueba si la etiqueta es una operación de suma, resta, multiplicación, división o un valor.
- Si es uno de los 4 primeros se introducen si respectivo signo dentro de un arreglo.
- Si es la etiqueta valor le asignamos al registro 'al' el valor de 1.

```
1  comparacion macro arregloAux, arreglo
2      Local suma, resta, multiplicacion, division, valor, salir
3
4      mov al, arregloAux[0]
5      cmp al, 'S'
6      je suma
7      cmp al, 'R'
8      je resta
9      cmp al, 'M'
10     je multiplicacion
11     cmp al, 'D'
12     je division
13     cmp al, 'V'
14     je valor
15     mov al, 0
16     jmp salir
17
18     suma:
19         mov al, 0
20         mov arreglo[di], 2bh
21         inc cont
22         jmp salir
23
24     resta:
25         mov al, 0
26         mov arreglo[di], 2dh
27         inc cont
28         jmp salir
29
30     multiplicacion:
31         mov al, 0
32         mov arreglo[di], 2ah
33         inc cont
34         jmp salir
35
36     division:
37         mov al, 0
38         mov arreglo[di], 2fh
39         inc cont
40         jmp salir
41
42     valor:
43         mov al, 1
44         jmp salir
45
46     salir:
47     endm
```

obtenerStringNumero

Concatena valor del número que viene dentro de las etiquetas de valor en un vector determinado.

- Compara si viene un '<', si es así quiere decir que ya terminó el número y empieza otra etiqueta.
- Sino es así guarda el carácter del número en el arreglo en la posición di.
- Al terminar el proceso de concatenación del número concatena un espacio y el signo de \$.

```
1  obtenerStringNumero macro arreglo, arregloAux
2      Local concatenar, salir
3
4      concatenar:
5          mov al, arreglo[si]
6          cmp al, '<'
7          je salir
8
9          mov arregloAux[di], al
10
11         inc contG
12         inc si
13         inc di
14         inc cont
15         jmp concatenar
16
17     salir:
18         mov al, 20h      ; ascii del signo dolar $
19         mov arregloAux[di], al
20         inc di
21         inc cont
22         mov al, 24h      ; ascii del signo dolar $
23         mov arregloAux[di], al
24     endm
```

Descripción Procedimientos

Solo existe un procedimiento el cual es el main, del cual existen varias etiquetas para poder hacer funcional el programa.

Etiqueta encabezado

Muestra el encabezado inicial, esto solo se mostrará al iniciar la aplicación.

```
1 encabezado:
2     print L_P0
3     print E_P1
4     print E_P2
5     print E_P3
6     print E_P4
7     print E_P5
8     print E_P6
9     print E_P7
10    print E_P8
11    print E_P9
12    print L_P0
13    print salto
14    jmp menu
```

```
1 E_P1 db 0ah, 0dh, '|   UNIVERSIDAD DE SAN CARLOS DE GUATEMALA   |', '$'
2 E_P2 db 0ah, 0dh, '|           FACULTAD DE INGENIERIA           |', '$'
3 E_P3 db 0ah, 0dh, '|           ESCUELA DE CIENCIAS Y SISTEMAS   |', '$'
4 E_P4 db 0ah, 0dh, '| ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1 |', '$'
5 E_P5 db 0ah, 0dh, '|           SECCION B                         |', '$'
6 E_P6 db 0ah, 0dh, '|           PRIMER SEMESTRE 2021             |', '$'
7 E_P7 db 0ah, 0dh, '|           -> Jose Ottoniel Sincal Guitzol <- |', '$'
8 E_P8 db 0ah, 0dh, '|           -> 201801375 <-                  |', '$'
9 E_P9 db 0ah, 0dh, '|           -> Primera Practica Assembler <- |', '$'
10 L_P0 db 0ah, 0dh, '|-----|', '$'
```

Etiqueta menú

Muestra el menú con las acciones que se podrán realizar en la aplicación.

- Pide que se ingrese un número por medio del macro getChar, el cual corresponde a la opción a realizar.
- Luego valida el número de entrada, si es una opción valida salta a una etiqueta correspondiente, de lo contrario saltará al menú.

```
1 M_P1 db 0ah, 0dh, '* ^^<      M E N U      P R I N C I P A L      >^^ *', '$'
2 M_P2 db 0ah, 0dh, '* -> 1. Cargar Archivo      >^^ *', '$'
3 M_P3 db 0ah, 0dh, '* -> 2. Calculadora        >^^ *', '$'
4 M_P4 db 0ah, 0dh, '* -> 3. Factorial          >^^ *', '$'
5 M_P5 db 0ah, 0dh, '* -> 4. Crear Reporte      >^^ *', '$'
6 M_P6 db 0ah, 0dh, '* -> 5. Salir              >^^ *', '$'
7 elegir db 0ah, 0dh, ' -> Seleccione una opcion: ', '$'
```

```

1  menu:
2      print salto
3      print L_P1
4      print M_P1
5      print M_P2
6      print M_P3
7      print M_P4
8      print M_P5
9      print M_P6
10     print L_P1
11     print elegir
12     getChar
13
14     cmp al, 31h
15     je cargarArchivo
16     cmp al, 32h
17     je calculadora
18     cmp al, 33h
19     je factorial
20     ;cmp al, 34h
21     ;je reporte
22     cmp al, 35h
23     je salir
24     jmp menu

```

Etiqueta cargarArchivo

Pide la ruta del archivo a cargar por medio del macro obtenerRuta. Para luego abrir el archivo y leer el contenido el cual guardaremos en un arreglo.

```

1  cargarArchivo:
2      print salto
3      print ingreseruta
4      print salto
5      limpiar ruta, SIZEOF ruta, 24h ;limpiamos el arreglo bufferentrada con $
6      obtenerRuta ruta ;obtenemos la ruta en buffer de entrada
7      abrir ruta, handlerentrada ;le mandamos la ruta y el handler, que será la referencia al fichero
8      limpiar info, SIZEOF info, 24h ;limpiamos la variable donde guardaremos los datos del archivo
9      leer handlerentrada, info, SIZEOF info ;leemos el archivo
10
11     print salto
12     print info
13     print salto

```

Etiqueta cerrarArchivo

Cierra el archivo que hemos abierto anteriormente.

```
1  cerrarArchivo:
2      cerrar handlerentrada
3      limpiar arregloOperacion, sizeof arregloOperacion, 24h
4      mov contG, 0
5      mov cont, 0
```

Etiqueta analizarArchivo

Analiza la información que se leyó y se guardó del archivo que cargamos.

- Llevaremos un conteo general para eso, el cual nos servirá para ir leyendo cada una de las posiciones del arreglo de la información de archivo.
- Pasamos el carácter contenido en la posición 'si' al registro 'al', validamos que no sea el final de la cadena para poder seguir validando. Si lo es imprime el resultado analizado.
- De lo contrario iremos a analizar la información para leer las etiquetas correspondientes de cada archivo y posteriormente comparar si es una suma, resta, multiplicación, división o un valor.
- Si es un valor saltamos a la etiqueta 'esNumero', sino seguimos analizando.

NOTA: esta etiqueta se complementa con los macros [analizar](#) y [comparacion](#), que ya mencionamos anteriormente.

```
1  analizarArchivo:
2      mov si, contG
3
4      mov al, info[si]
5      cmp al, '$'
6      je resultadoArchivo
7
8      limpiar arregloAux, sizeof arregloAux, 24h
9      analizar info, arregloAux
10
11     mov di, cont
12     comparacion arregloAux, arregloOperacion
13     mov num, al
14     cmp num, 1
15     je esNumero
16     jmp analizarArchivo
```

Etiqueta esNumero

Concatena el número que viene dentro de las etiquetas de <valor> NUM </valor>, del archivo cargado.

- Esta etiqueta se complementa con el macro [obtenerStringNumero](#) definido anteriormente.

```
1  esNumero:
2      limpiar arregloAux, SIZEOF arregloAux, 24h
3      mov si, contG
4      mov di, cont
5      obtenerStringNumero info, arregloOperacion
6      jmp analizarArchivo
```

Etiqueta resultadoArchivo

Muestra el resultado final del archivo de carga.

```
1  resultadoArchivo:
2      inc di
3      inc cont
4      mov al, 24h      ; ascii del signo dolar $
5      mov arregloOperacion[di], al
6      print arregloOperacion
7      limpiar arregloOperacion, SIZEOF arregloOperacion, 24h
8      jmp menu
```

Etiqueta calculadora

Pide un numero el cual va acumulando, luego pide un operador para saber qué operación se realizará. Se verifica el operador para saber a qué etiqueta saltará para realizar una operación determinada.

```
1  calculadora:
2      print salto
3      print ingresarNum
4      obtenerNumero num, diez
5      mov al, num
6      mov resultado, al
7      print ingresarOp1
8      getChar
9      cmp al, '+'
10     je sumar
11     cmp al, '-'
12     je restar
13     cmp al, '*'
14     je multiplicar
15     cmp al, '/'
16     je dividir
17     jmp calculadora
```

Etiqueta sumar, restar, multiplicar y dividir

Pide otro número con el cual se operará con el que guardamos anteriormente.

```
1  sumar:
2      print ingresarNum
3      obtenerNumero num, diez
4      mov al, resultado
5      add al, num
6      mov resultado, al
7      jmp pedirMas
```

```
1  restar:
2      print ingresarNum
3      obtenerNumero num, diez
4      mov al, resultado
5      sub al, num
6      mov resultado, al
7      jmp pedirMas
```



```
1  multiplicar:
2      print ingresarNum
3      obtenerNumero num, diez
4      mov al, resultado
5      mov bl, num
6      imul bl
7      mov resultado, al
8      jmp pedirMas
```



```
1  dividir:
2      print ingresarNum
3      obtenerNumero num, diez
4      mov al, resultado
5      mov bl, num
6      cbw
7      idiv bl
8      mov resultado, al
9      jmp pedirMas
```

Etiqueta pedirMas

Pide nuevamente otro operador para seguir haciendo operaciones o simplemente podemos terminar el proceso de la calculadora.



```
1  pedirMas:
2      print ingresarOp2
3      getChar
4      cmp al, '+'
5      je sumar
6      cmp al, '-'
7      je restar
8      cmp al, '*'
9      je multiplicar
10     cmp al, '/'
11     je dividir
12     cmp al, ';'
13     je mostrarResultado
14     jmp pedirMas
```


Etiqueta mostrarResultado

Muestra el resultado de las operaciones pedidas o ingresadas.

```
1  mostrarResultado:
2      print salto
3      print msjSal
4      printNum resultado, residuo, diez
5      print salto
6      jmp menu
```

Etiqueta factorial

Manda a llamar el resultado del factorial de un cierto número.

- Primero pide el número del cual deseamos saber el factorial.
- Luego iniciamos una variable en 0, el cual será el inicio de nuestras operaciones del factorial.
- Cuando la variable es mayor al número del que deseamos saber el factorial para y muestra el resultado final junto con las operaciones realizadas.

```
1  factorial:
2      print salto
3      print ingresarNum
4
5      mov ah, 01h
6      int 21h
7      sub al, 30h
8
9      mov limite, al
10
11     print salto
12     print msjOpe
13
14     mov resf, 0
15     operaciones:
16         printNum resf, residuo, diez
17         printFacEqual
18
19         mov al, resf
20         mov num, al
21
22         obtenerFactorialAux num
23         printNum num, residuo, diez
24         printPtcEsp
25
26         inc resf
27         mov al, limite
28         cmp resf, al
29         jle operaciones
30
31
32         sub resf, 1
33         obtenerFactorialAux resf
34         print msjSal
35         printNum resf, residuo, diez
36         print salto
37         jmp menu
```