Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Arquitectura de computadores y ensambladores 1
Primer semestre 2022

MANUAL TÉCNICO

201800937
José Abraham Solórzano Herrera

En el segmento se importa el archivo de macos.

```
; ************************** [SEGMENTO][LIBS] ***************************
include Macros.asm

.model small

; ************************** [SEGMENTO][STACK] **************************
.stack

; ************************** [SEGMENTO][DATA] ***************************
.data
```

En el segmento de Data, se declaran todas las variables a utilizar, además de incluir los procs

```
.data

; ************************ DECLARACION DE VARIABLES ************************
; ************** [IDENTIFICADOR] **************
_salto          db 0ah,0dh,             "$"
_opcion         db 0ah,0dh,             "> Ingrese Opcion: $"
_regresar       db 0ah,0dh,             "1. regresar$"
_num            db                      "> ", "$"
_resultado      db 0ah,0dh,             "> resultado es ", "$"
; ************** [ERRORES] **************
_error1         db 0ah,0dh,             "> Error al Abrir Archivo, no Existe ",   "$"
_error2         db 0ah,0dh,             "> Error al Cerrar Archivo",              "$"
_error3         db 0ah,0dh,             "> Error al Escribir el Archivo",         "$"
_error4         db 0ah,0dh,             "> Error al Crear el Archivo",            "$"
_error5         db 0ah,0dh,             "> Error al Leer al Archivo",             "$"
_error6         db 0ah,0dh,             "> Error en el Archivo",                  "$"
; ************** [IDENTIFICADOR] **************
_cadena1        db 0ah,0dh,             "Universidad de San Carlos de Guatemala$"
_cadena2        db 0ah,0dh,             "Facultad de Ingenieria$"
_cadena3        db 0ah,0dh,             "Escuela de Ciencias y Sistemas$"
_cadena4        db 0ah,0dh,             "Arquitectura de Compiladores y ensambladores 1$"
_cadena5        db 0ah,0dh,             "Seccion <A>$"
_cadena6        db 0ah,0dh,             "<Jose Abraham Solorzano Herrera>$"
_cadena7        db 0ah,0dh,             "<201800937>$"
```

```
; ************************************ [PROCS] ************************************

; ************************** [IDENTIFICADOR] **************************
identificador proc far
    GetPrint _salto
    GetPrint _cadena1
    GetPrint _cadena2
    GetPrint _cadena3
    GetPrint _cadena4
    GetPrint _cadena5
    GetPrint _cadena6
    GetPrint _cadena7
    ret
identificador endp
```

En la parte del Archivo Macos.asm se utilizó una función que sirve para mostrar el contenido en dx.

```
; ***************************** [MACROS] ******
; ************** [IMPRIMIR] **************
GetPrint macro buffer
  MOV AX,@data
  MOV DS,AX
  MOV AH,09H
  MOV DX,OFFSET buffer
  INT 21H
endm
```

Se crearon tres macros que sirven para obtener un char, donde es guardada en al, el segundo y tercer macro sirve para poder guardar una cadena de 10 bytes como máximo.

```
; ************** [CAPTURAR ENTRADA] **************
GetInput macro
  MOV AH,01H
  int 21H
endm

GetInputMax macro _resultS
  mov ah, 3fh           ; int21 para leer fichero o dispositivo
  mov bx, 00            ; handel para leer el teclado
  mov cx, 10            ; bytes a leer (aca las definimos con 10)
  mov dx, offset[_resultS]
  int 21h
endm
```

```
; ************** [VARIABLE][GET] **************
GetText macro buffer
  LOCAL Ltext, Lout
  xor si,si  ; Es igual a mov si,0

  Ltext:
    GetInput
    cmp al,0DH ; Codigo ASCCI [\n -> Hexadecimal]
    je Lout
    mov buffer[si],al ; mov destino,fuente
    inc si ; si = si + 1
    jmp Ltext

  Lout:
    mov al,24H ; Codigo ASCCI [$ -> Hexadecimal]
    mov buffer[si],al
endm
```

Se creo una función GetRoot que nos sirve para obtener un buffer que nos va a servir para la ruta, está termina en nul que se expresa como 00h, a diferencia de GetText que termina en $.

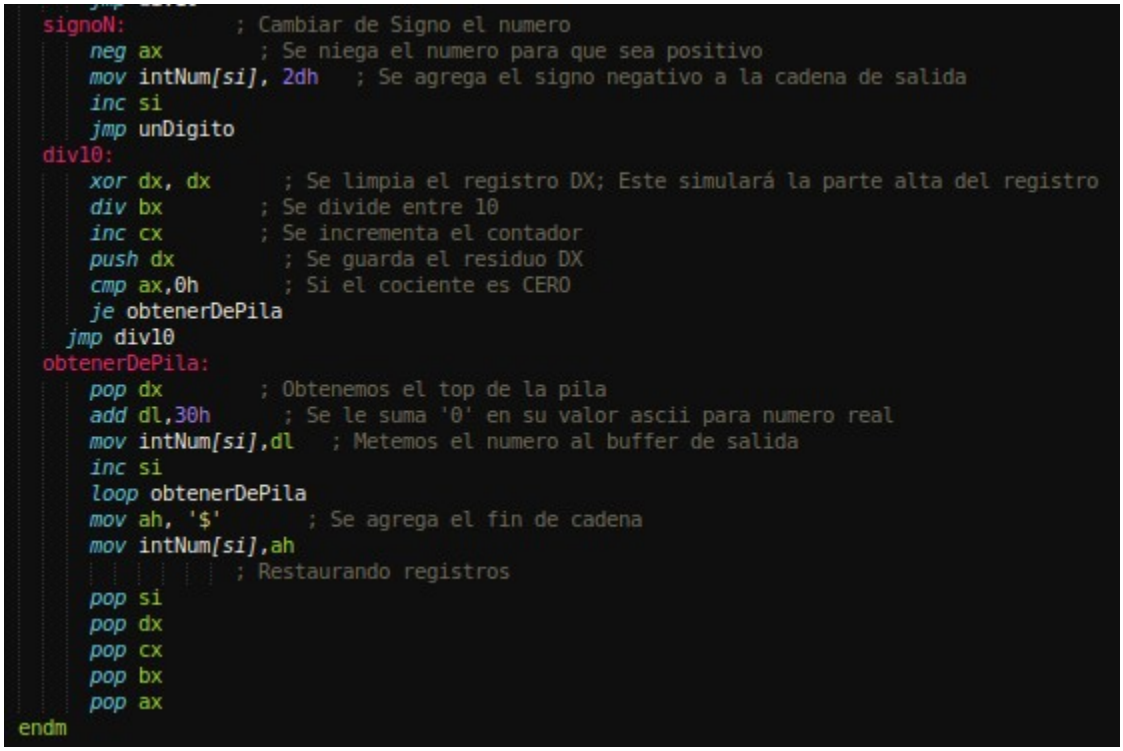
```
; ************** [VARIABLE][GET] **************
GetText macro buffer
  LOCAL Ltext, Lout
  xor si,si  ; Es igual a mov si,0

  Ltext:
    GetInput
    cmp al,0DH ; Codigo ASCCI [\n -> Hexadecimal]
    je Lout
    mov buffer[si],al ; mov destino,fuente
    inc si ; si = si + 1
    jmp Ltext

  Lout:
    mov al,24H ; Codigo ASCCI [$ -> Hexadecimal]
    mov buffer[si],al
endm
```

Se creo GetOpenFile para poder abrir el buffer que se obtuvo con GetRoot, ademas se creo el macro GetCloseFile, para finalizar el archivo; por útlimo se creo el macro

```
; ************** [PATH][OPEN] *************
GetOpenFile macro buffer,handler
  mov ah,3dh
  mov al,02h
  lea dx,buffer
  int 21h
  jc Lerror1
  mov handler,ax
endm
; ************** [PATH][CLOSE] *************
GetCloseFile macro handler
  mov ah,3eh
  mov bx,handler
  int 21h
  jc Lerror2
endm
; ************** [PATH][READ] *************
GetReadFile macro handler,buffer,numbytes
  mov ah,3fh
  mov bx,handler
  mov cx,numbytes
  lea dx,buffer
  int 21h
  jc Lerror5
endm
```

Convertir int a string, que queda almacenado en ax

```
;convierte un NUMERO a CADENA que esta guardado en AX
Int_String macro intNum
  local div10, signoN, unDigito, obtenerDePila
  ;Realizando backup de los registros BX, CX, SI
  push ax
  push bx
  push cx
  push dx
  push si
  xor si,si
  xor cx,cx
  xor bx,bx
  xor dx,dx
  mov bx,0ah        ; Divisor: 10
  test ax,1000000000000000  ; veritficar si es numero negativo (16 bits)
  jnz signoN
  unDigito:
      cmp ax, 0009h
      ja div10
      mov intNum[si], 30h   ; Se agrega un CERO para que sea un numero de dos digitos
      inc si
      jmp div10
```



```
signoN:         ; Cambiar de Signo el numero
    neg ax        ; Se niega el numero para que sea positivo
    mov intNum[si], 2dh   ; Se agrega el signo negativo a la cadena de salida
    inc si
    jmp unDigito
divl0:
    xor dx, dx      ; Se limpia el registro DX; Este simulará la parte alta del registro
    div bx        ; Se divide entre 10
    inc cx        ; Se incrementa el contador
    push dx         ; Se guarda el residuo DX
    cmp ax,0h       ; Si el cociente es CERO
    je obtenerDePila
  jmp divl0
obtenerDePila:
    pop dx        ; Obtenemos el top de la pila
    add dl,30h      ; Se le suma '0' en su valor ascii para numero real
    mov intNum[si],dl   ; Metemos el numero al buffer de salida
    inc si
    loop obtenerDePila
    mov ah, '$'       ; Se agrega el fin de cadena
    mov intNum[si],ah
                ; Restaurando registros
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
endm
```

De igual forma para un string a int, qu queda almacenado en ax

```
;convierte una CADENA A NUMERO, este es guardado en AX.
String_Int macro stringNum
  local ciclo, salida, verificarNegativo, negacionRes
  push bx
  push cx
  push dx
  push si
  ;Limpiando los registros AX, BX, CX, SI
  xor ax, ax
  xor bx, bx
  xor dx, dx
  xor si, si
  mov bx, 000Ah           ;multiplicador 10
  ciclo:
      mov cl, stringNum[si]
      inc si
      cmp cl, 2Dh           ; compara para ignorar el "-"
      jz ciclo              ; Se ignora el simbolo '-' de la cadena
      cmp cl, 30h           ; Si el caracter es menor a '0', implica que es negativo (verificacion)
      jb verificarNegativo      ; ir para cuando es un negativo
      cmp cl, 39h           ; Si el caracter es mayor a '9', implica que es negativo (verificacion)
      ja verificarNegativo
      sub cl, 30h         ; Se le resta el ascii '0' para obtener el número real
      mul bx                ; multplicar ax
      mov ch, 00h
      add ax, cx            ; sumar para obtener el numero total
      jmp ciclo
  negacionRes:
      neg ax            ; negacion por si es negativo el resultado
      jmp salida
  verificarNegativo:
      cmp stringNum[0], 2Dh     ; Si existe un signo al inicio del numero, negamos el numero
      jz negacionRes
  salida:
                        ; Restaurando los registros AX, BX, CX, SI
      pop si
      pop dx
      pop cx
      pop bx
endm
```

El macro nos sirve para solicitar un número, que va a ser guardado en un string, para luego ser convertido en int.

```
; recibe una cadena y lo convierte en numero
Solicitar_Numero macro cadenaNumero, numeroConvertido
  mov ah, 3fh           ; int21 para leer fichero o dispositivo
  mov bx, 00            ; handel para leer el teclado
  mov cx, 10            ; bytes a leer (aca las definimos con 10)
  mov dx, offset[cadenaNumero]
  int 21h
  GetAC cadenaNumero
  cmp cadenaNumero,65h  ; Codigo ASCCI [e -> Hexadecimal] salir del programa
  je Lmenu

  ; Convertimos la cadena a numero es guardado en AX
  String_Int cadenaNumero
  mov numeroConvertido, ax    ; Guardar en el "int"
endm

;limpiar variables
GetClean macro _numero1, _numero2, _calcuResultado
    mov _numero1, 0
    mov _numero1, ax
    mov _calcuResultado, 0
    mov _numero2, 0
endm
```

Se inicia con al etiqueta Linicio donde muestra los datos del curso, tanto como del propietario; luego a dar enter apararece la nueva etiqueta que accede a la calculadora, o si deseamos cargar un archivo o hasta salir.

```
Linicio:
    ; LLamamos identificador
    call identificador

    GetInput
    cmp al,0Dh ; Codigo ASCCI [Enter -> Hexadecimal]
    je Lmenu
    jmp Linicio
                    [MENU]

Lmenu:
    ; LLamamos menu
    call funcMenu

    ; Obtenemos el Caracter
    GetInput

    cmp al,31H ; Codigo ASCCI [1 -> Hexadecimal]
    je Loperacion
    cmp al,32H ; Codigo ASCCI [2 -> Hexadecimal]
    je Lfile
    cmp al,33H ; Codigo ASCCI [3 -> Hexadecimal]
    je Lsalir
    jmp Lmenu
```

Para el menu del archivo tenemos la primera etiquet que nos sirve si queremos seleccionar el path, cerrar o mostrar el contenido.

```
LreadFile:
    GetPrint    _salto
    GetPrint    _cadena16                                   ; Ingreso de Path
    ; GetInput
    GetRoot     _bufferInput                                ; Capturar Path
    GetOpenFile _bufferInput,_handleInput                   ; Abrir file
    GetReadFile _handleInput,_bufferInfo,SIZEOF _bufferInfo ; Guardar Contenido
    GetInput
    jmp Lfile
```

```
LmostrarFile:
    GetPrint _salto
    GetPrint _bufferInfo
    GetPrint _salto
    GetInput

    jmp Lfile

LcloseFile:
    GetCloseFile _handleInput
    GetInput
    jmp Lfile
```

Luego tenemos la etiqueta Loperacion, que nos sirve para interactuar con la calculadora, donde como primer punto, vamos a obtener el primer digito, luego el segundo digito. Por último la operación a realizar.

```
Loperacion:
    call funcCalculadora ; LLamamos calculadora
    GetClean _numero1, _numero1, _calcuResultado
    GetPrint _num ; Obtenemos el primer número
    Solicitar_Numero _numero1S, _numero1


    GetPrint _num ; Obtenemos el segundo número
    Solicitar_Numero _numero2S, _numero2

    GetPrint _num
    GetInputMax _resultS

    cmp _resultS,2BH ; Codigo ASCCI [+ -> Hexadecimal]
    je Lsuma
    cmp _resultS,2DH ; Codigo ASCCI [- -> Hexadecimal]
    je Lresta
    cmp _resultS,78H ; Codigo ASCCI [x -> Hexadecimal]
    je Lmultiplicacion
    cmp _resultS,2FH ; Codigo ASCCI [/ -> Hexadecimal]
    je Ldivision
    cmp _resultS,5EH ; Codigo ASCCI [^ -> Hexadecimal]
    je Lpotencia
    cmp _resultS,65h  ; Codigo ASCCI [e -> Hexadecimal] salir del programa
    je Lmenu
```

Operación suma, que se encuentra en la etiqueta Lsuma, por medio de un add.

```
Lsuma: ; operación suma

    mov dx, _numero1
    add dx, _numero2
    mov _calcuResultado, dx
    mov ax, _calcuResultado
    Int_String _numero1S ; convierte el numero guardado en ax
    GetPrint _resultado
    GetPrint _numero1S

    ;GetClean _numero1, _numero1, _calcuResultado

    jmp Loperacion2
```

La etiqueta Lresta, donde opera la resta, por medio de un sub.

```
Lresta: ; operación resta

    mov dx, _numero1
    sub dx, _numero2
    mov _calcuResultado, dx
    mov ax, _calcuResultado
    Int_String _numero1S ; convierte el numero guardado en ax
    GetPrint _resultado
    GetPrint _numero1S

    jmp Loperacion2
```

La etiqueta Lmultiplicacion, que es el encargao de realizar la multiplicación, por medio de imul.

```
Lmultiplicacion: ; operacion multiplicacion

    mov ax, _numero1
    mov bx, _numero2
    imul bx
    mov _calcuResultado,ax
    mov ax,_calcuResultado
    Int_String _numero1S ; convierte el numero guardado en ax
    GetPrint _resultado
    GetPrint _numero1S
```

La etiqueta Ldivision, que es el encargado de realizar la división, por medio de cwd.

```
Ldivision:

    mov ax,_numero1
    cwd                 ; Convertimos a dobleword
    mov bx,_numero2
    idiv bx             ; ax/bx ax = Resultado

    mov _calcuResultado,ax  ; guardamos en calcuIN1
    Int_String _numero1S ; convierte el numero guardado en ax
    GetPrint _resultado
    GetPrint _numero1S
    jmp Loperacion2
```

Par la potencia se crea una macro donde vamos a recorrer hasta el máximo del exponente, donde se multiplicara en cada caso.

```
GetPotencia macro _result, _num1S, _num1I, _num2I, _numTemp
  LOCAL _Lout, _L0, _Lpote
  push ax
  push cx
  push bx
  push dx
  xor SI, SI ; contador para el contenedor
  xor DI, DI ; contador para posiciones
  inc SI ; incremento 1° vez
  inc SI ; incremento 2° vez
  mov ax,_num1I
  mov _numTemp, ax
  cmp _num2I,0
  je _L0
  jmp _Lpote

  _L0:

    mov _num1I,1
    mov _num1S,'1'
    jmp _Lout

  _Lpote:
    mov ax, _numTemp
    mov bx, _num1I
    imul bx
    mov _result,ax
    mov ax,_result
    Int_String _num1S ; convierte el numero guardado en ax
    mov _numTemp,ax
    cmp SI,_num2I
    je _Lout
    inc SI
    jmp _Lpote

  _Lout:
    GetPrint _resultado
    GetPrint _num1S

endm
```