Universidad De San Carlos de Guatemala   
Facultad de Ingeniería  
Escuela de Ciencias y Sistemas

Laboratorio Arquitectura de computadores y ensambladores 1  
Sección “N”

**“MANUAL TÉCNICO”**

Diego André Mazariegos Barrientos

Carnet: 202003975

**Objetivos**

General:

Brindar al lector una guía que contenga la información del manejo de clases, atributos, métodos y del desarrollo de la interfaz gráfica para facilitar futuras actualizaciones y futuras modificaciones realizadas por terceros.

Específicos:

* Mostrar al lector una descripción lo más completa y detallada posible del SO, IDE entre otros utilizados para el desarrollo de la aplicación.
* Proporcionar al lector una concepción y explicación técnica - formal de los procesos y relaciones entre métodos y atributos que conforman la parte operativa de la aplicación.

**Introducción**

Este manual técnico tiene como finalidad dar a conocer al lector que pueda requerir hacer modificaciones futuras al software el desarrollo de la aplicación denominada “PROYECTO FASE 1” desarrollada durante el transcurso de la tercera semana de diciembre del año 2022, indicando el programa utilizado para su creación, su versión, requerimientos del sistema, etc…

**Descripción de la Solución**

Para poder desarrollar este proyecto se analizó los requisitos solicitados, sus restricciones tanto humanas y de equipo del proyecto; tanto así como el ambiente y forma de trabajo de los futuros operadores de la aplicación.

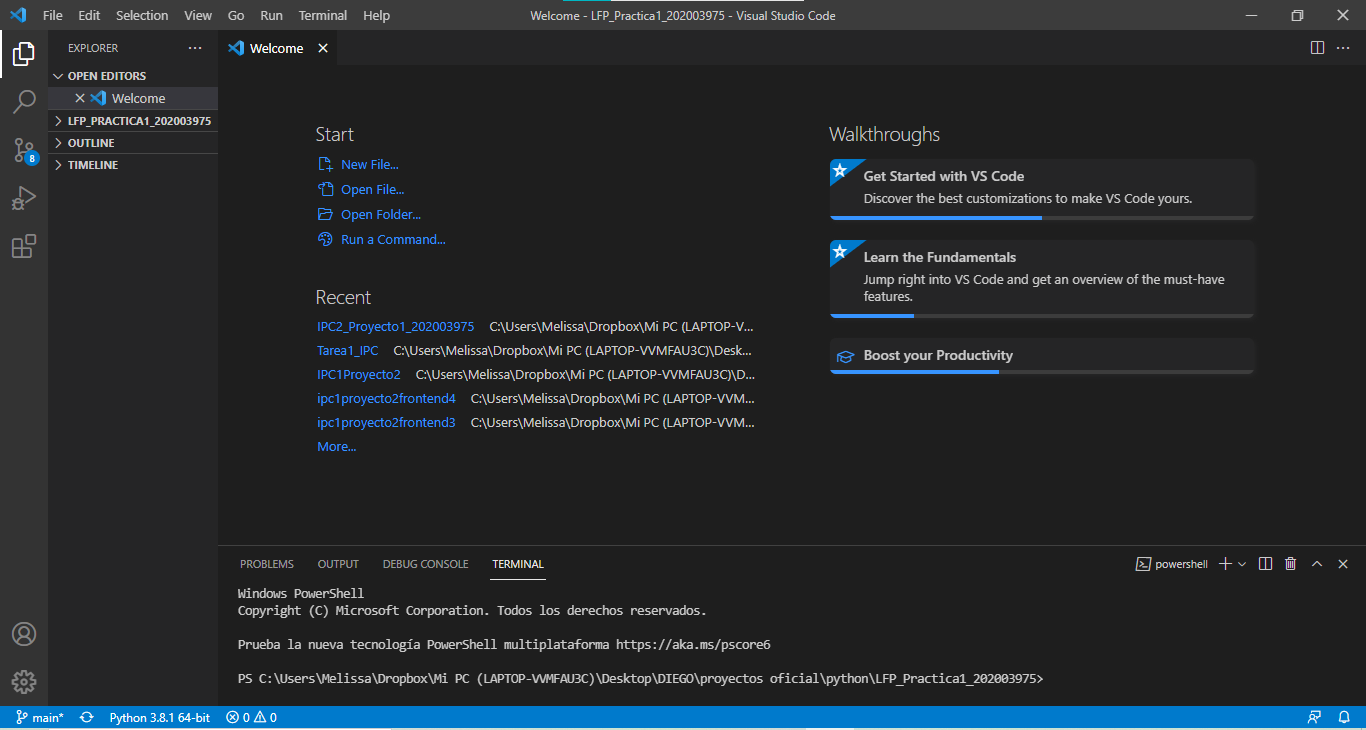
Entre las consideraciones encontramos con mayor prioridad están:

* Realizar la lectura de las operaciones de entrada con el formato correcto.
* Análisis completo de la entrada una vez verificada que la entrada sea correcta procediendo a elaborar las respectivas funcionalidades dependiendo de las acciones que tome el usuario.

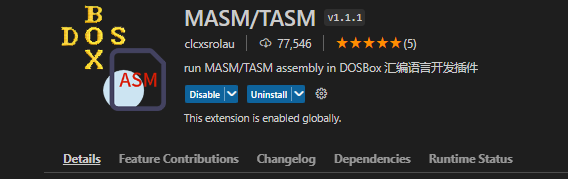
* Presentación de la interfaz de forma agradable y fácil de usar.

**IDE**

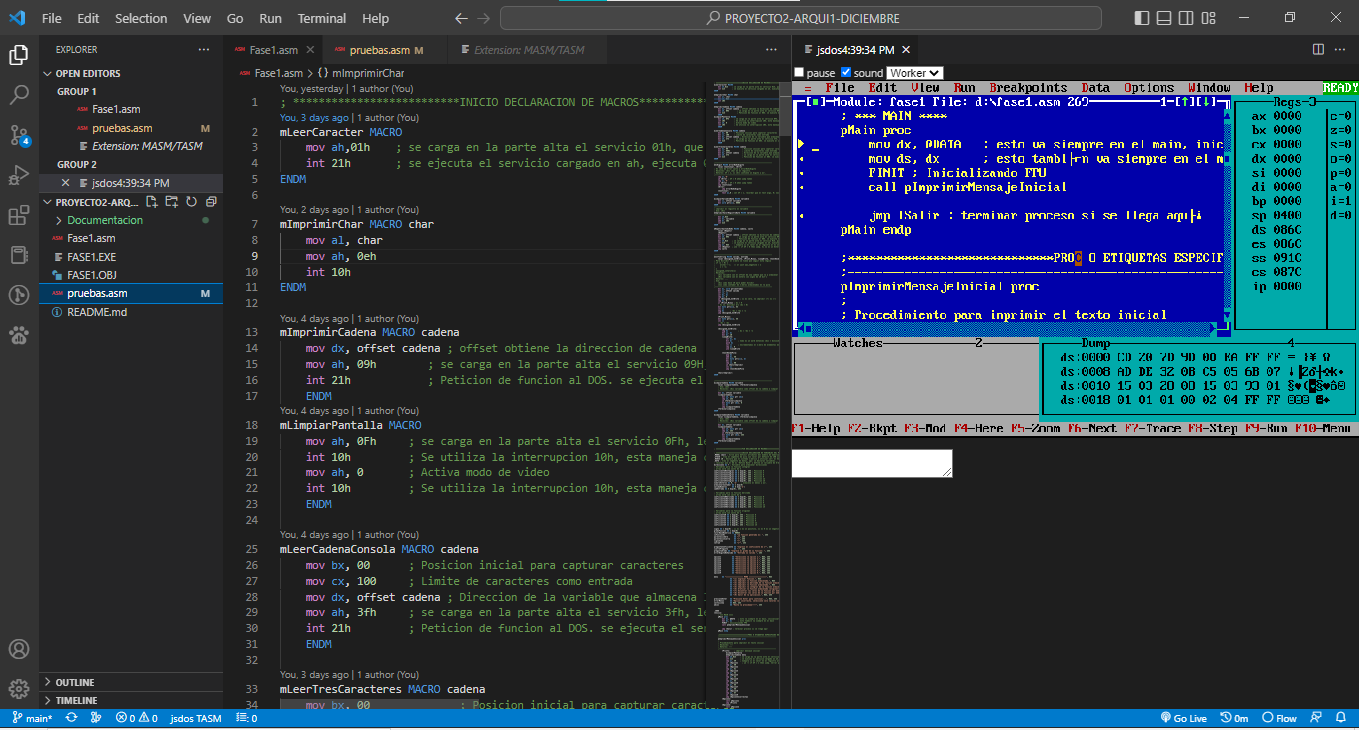
El IDE con el que se desarrolló el proyecto “PROYECTO FASE 1” fue Visual Studio Code, debido a su apoyo al desarrollador gracias a su asistente que detecta errores semánticos, sintácticos del código por lo cual ayudan y hacen que la duración de la fase de programación sea más corta, además posee una interfaz muy agradable y fácil de entender en el modo debugging.



Todo esto a través de la extensión MASM/TASM que es una extensión libre y gratuita para programar en el lenguaje ensamblador específicamente en MASM y TASM que incluye un depurador muy ordenado e eficiente con el cual se elaboro este proyecto.



Depurador utilizado simulando jsdos en visual studio.



**Requerimientos de IDE:**

* **Hardware**

Visual Studio Code es una pequeña descarga (<200 MB) y ocupa un espacio en disco de <500 MB. VS Code es liviano.

Se recomienda:

Procesador de 1,6 GHz o más rápido.

1 GB de RAM.

* **Software**
* OS X El Capitan (10.11+).
* Windows 7 (con .NET Framework 4.5.2), 8.0, 8.1 y 10 (32 y 64 bits).
* Linux (Debian): Ubuntu Desktop 16.04, Debian 9.
* Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 8, Fedora 24.

**SIMULADOR DOSBox**

DOSBox es un emulador de sistema operativo que permite que las computadoras modernas ejecuten aplicaciones y juegos diseñados para el sistema operativo MS-DOS. Este programa es útil para los usuarios que quieren jugar juegos antiguos o usar aplicaciones solo para MS-DOS, pero no tienen acceso a computadoras con sistemas operativos más antiguos. Para ejecutar DOSBox necesita las siguientes especificaciones mínimas:

Sistemas operativos: sistemas operativos modernos como Windows, macOS y Linux.

CPU: cualquier CPU que admita instrucciones x86 o x86-64.

Memoria RAM: 256 MB o más.

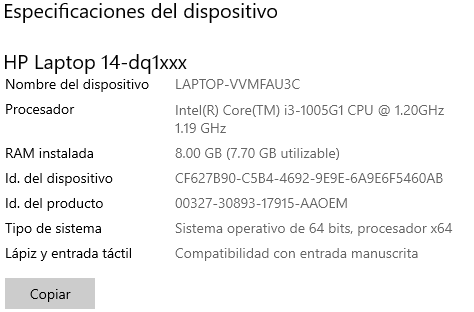
Espacio en disco: 20 MB o más.

Tenga en cuenta que estas son las especificaciones mínimas para ejecutar DOSBox. Si usa DOSBox para jugar o usar aplicaciones que requieren un mayor rendimiento, es posible que necesite especificaciones más altas.

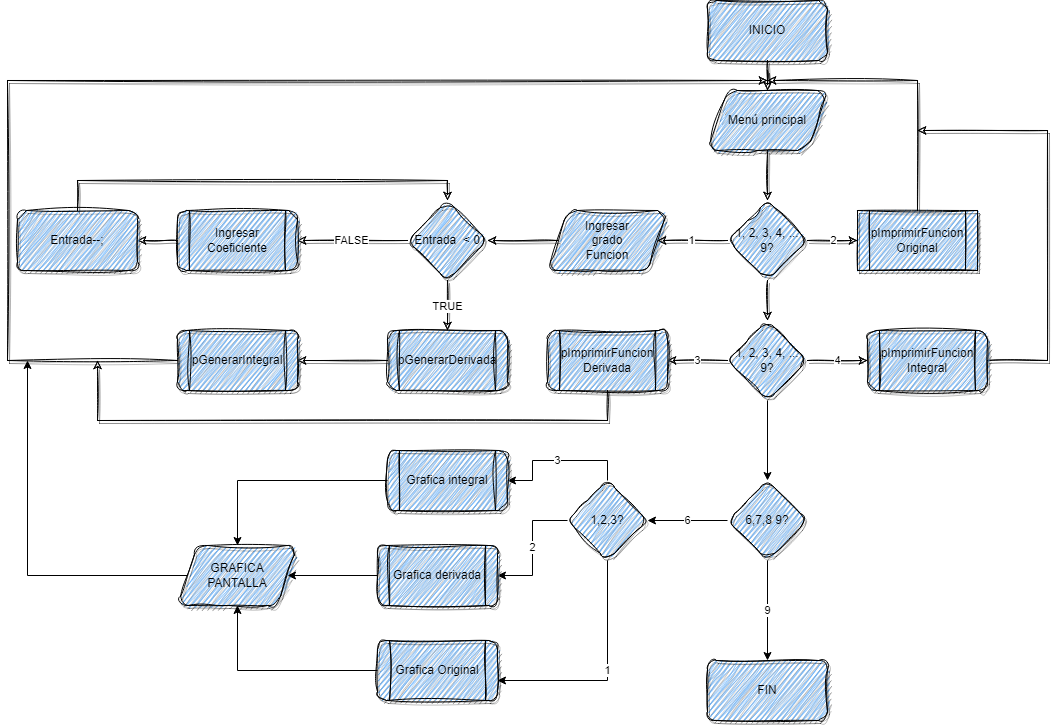
**Requisitos del programa**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sistema operativo** | **Memoria RAM mínima** | **Memoria RAM recomendada** | **Espacio en disco mínimo** | **Espacio en disco recomendado** |
| El programa puede ser instalado en cualquier sistema operativo, siempre y cuando se utilice de DOSBox para ejecutar este. | 500 MB | 8 GB | 7 KB | 1 GB |

**Máquina en la cual fue desarrollado el programa**



**Diagrama flujo**



**CÓDIGO COMENTADO**

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INICIO DECLARACION DE MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mLeerCaracter MACRO

    mov ah,01h    ; se carga en la parte alta el servicio 01h, que lee un caracter de la entrada y lo guarda en el registro al.

    int 21h       ; se ejecuta el servicio cargado en ah, ejecuta 01h.

ENDM

mImprimirChar MACRO char

    mov al, char

    mov ah, 0eh

    int 10h

ENDM

mImprimirCadena MACRO cadena

    mov dx, offset cadena ; offset obtiene la direccion de cadena

    mov ah, 09h        ; se carga en la parte alta el servicio 09H, el cual despliega una cadena, que es imprimir n columnas hacia adelante.

    int 21h            ; Peticion de funcion al DOS. se ejecuta el servicio cargado en ah. Ejecutar funcion 09,

    ENDM

mLimpiarPantalla MACRO

    mov ah, 0Fh     ; se carga en la parte alta el servicio 0Fh, lee el modo actual de video.

    int 10h         ; Se utiliza la interrupcion 10h, esta maneja casi todos los servicios de la pantalla; Video Service.

    mov ah, 0       ; Activa modo de video

    int 10h         ; Se utiliza la interrupcion 10h, esta maneja casi todos los servicios de la pantalla; Video Service.

    ENDM

mLeerCadenaConsola MACRO cadena

    mov bx, 00      ; Posicion inicial para capturar caracteres

    mov cx, 100     ; Limite de caracteres como entrada

    mov dx, offset cadena ; Direccion de la variable que almacena los datos de entrada

    mov ah, 3fh     ; se carga en la parte alta el servicio 3fh, lee los datos de la consola

    int 21h         ; Peticion de funcion al DOS. se ejecuta el servicio cargado en ah. Ejecutar funcion 09,

    ENDM

mLeerTresCaracteres MACRO cadena

    mov bx, 00              ; Posicion inicial para capturar caracteres

    mov cx, 0002            ; Limite de caracteres como entrada

    mov dx, offset cadena   ; Direccion de la variable que almacena los datos de entrada

    mov ah, 3fh             ; se carga en la parte alta el servicio 3fh, lee los datos de la consola

    int 21h                 ; Peticion de funcion al DOS. se ejecuta el servicio cargado en ah. Ejecutar funcion 09,

ENDM

;---------------------------------------------------------

mIsDigit MACRO errorNoEsDigito

    local lContinuar, lError

; En caso de error salta a errorNoEsDigito

; Receives: [Al] char en el registro.

; Returns: ZF = 1, si [Al] contiene un digito y así.

;---------------------------------------------------------

    cmp al,'0'

    jb lError ; ZF = 0 when jump taken

    cmp al,'9'

    ja lError ; ZF = 0 when jump taken

    jmp lContinuar

    lError:

        jmp errorNoEsDigito

    lContinuar:

        test ax,0 ; set ZF = 1, recordar que el test algo, 0; nos garantiza retornar ZF = 1

ENDM

mLimpiarVariableByte MACRO variable

    mov si, offset variable

    mov word ptr[si], 0000

ENDM

;---------------------------------------------------------

; imprimir el registro en variable

; Use: [Dl]

mImprimirValorRegistroByte MACRO variable

;---------------------------------------------------------

    mov ah,02h

    mov dl, variable

    add dl, 30h

    int 21h

ENDM

mRepetirSaltoSiNoEs MACRO cadena, salto

    local lRepetir

    lRepetir:

    mov dx, offset cadena ; offset obtiene la direccion de cadena

    mov ah, 09h        ; se carga en la parte alta el servicio 09H, el cual despliega una cadena, que es imprimir n columnas hacia adelante.

    int 21h            ; Peticion de funcion al DOS. se ejecuta el servicio cargado en ah. Ejecutar funcion 09,

    mov ah,01h    ; se carga en la parte alta el servicio 01h, que lee un caracter de la entrada y lo guarda en el registro al.

    int 21h       ; se ejecuta el servicio cargado en ah, ejecuta 01h.

    cmp al, 0dh     ; Compara si el valor en el registro al es un Enter. al = Enter y 0dh = Enter, entonces ZF = 1, de lo contrario ZF = 0.

    jne lRepetir  ; jne -> if ZF = 0 then jump. Si no es un Enter salta a la etiqueta lPrint1.

    jmp salto

ENDM

;---------------------------------------------------------

mIntToString MACRO salida, entrada

    local lUnsigned\_IntWrite, lPrint\_Minus, lLoopWrite, lConvDesdePila, lSalirImprimir

; Para Escribir el número se tiene que seguir esta lógica

; if (x < 0) {

;     write('-');   // or just was\_negative = 1

;     x = -x;

; }

; unsigned\_intwrite(x)

; Receives:

;   [Si] variable con el offset de una cadena que va a almacenar el resultado

;   [Bx] variable con el entero con signo de 16 bit's

; Returns: ...

; Use:

;   [Di] como base 10 para poder dividir

;   [Cx] como contador de números almacenados en la pila

;---------------------------------------------------------

    mov bx, word ptr[entrada]

    mov si, offset salida

    xor cx, cx

    xor di, di

    cmp bx, 0

    je lUnsigned\_IntWrite ; si es cero, no imprimir (+) ni (-)

    cmp bx, 0

    jl lPrint\_Minus ; bx < 0

    ; Si no, Escribir más (bx > 0)

    mov byte ptr[si], 43

    inc si

    neg bx      ; -bx = +bx \* -1

    jmp lUnsigned\_IntWrite

    lPrint\_Minus:

    mov byte ptr[si], 45

    inc si

    ; -bx = -bx

    jmp lUnsigned\_IntWrite

    lUnsigned\_IntWrite:

        neg bx          ; bx = -bx \* -1

        mov ax, bx

        mov di, 10

        lLoopWrite:

            xor dx, dx

            div di      ; Como es un word entonces [Ax] = división, [Dx] = resto

            push dx

            inc cx      ; Incrementamos el número de elementos en la pila

            cmp ax, 0

            jne lLoopWrite

        lConvDesdePila:

            pop dx

            add dx, 48

            mov byte ptr[si], dl

            dec cx

            cmp cx, 00

            je lSalirImprimir

            inc si

            jmp lConvDesdePila

    lSalirImprimir:

ENDM

;---------------------------------------------------------

mLimpiarCadena MACRO variable

    local lLimpiarCadena, lTerminarLimpieza

    ; Use: [Ah]

    ; Receives: [Bx] variable como offset de la cadena a limpiar

    ;---------------------------------------------------------

    mov bx, offset variable

    lLimpiarCadena:

        mov ah, byte ptr [bx]

        cmp ah, 24h

        je lTerminarLimpieza

        mov byte ptr [bx], 0

        inc bx

        jmp lLimpiarCadena

    lTerminarLimpieza:

ENDM

mLimpiarCadenaEntero MACRO variable

    local lLimpiarCadena, lTerminarLimpieza

    ; Use: [Ah]

    ; Receives: [Bx] variable como offset de la cadena a limpiar

    ;---------------------------------------------------------

    mov bx, offset variable

    lLimpiarCadena:

        mov ah, byte ptr [bx]

        cmp ah, 24h

        je lTerminarLimpieza

        mov byte ptr [bx], 24h

        inc bx

        jmp lLimpiarCadena

    lTerminarLimpieza:

ENDM

; MACROS FASE 2

mActivarModoVideo MACRO

    push ax             ; extrayendo el valor de ax para no perderlo cuando se use la macro

    mov ax, 0013h       ; servicio requerido 13h

    int 10h

    mov ax, 0A000h      ; Nos posicionamos en la direccion de las variables del modo vídeo

    mov es, ax

    pop ax

ENDM

mDesactivarModoVideo MACRO

    push ax

    mov ax, 0003h       ; servicio requerido 0003h

    int 10h

    pop ax

ENDM

mDibujarEjeY macro

LOCAL LOOP\_I

    push dx

    push di

    mov dl, 1d

    mov di, 159d

    LOOP\_I:

        mov es:[di], dl

        add di, 320d         ; ubicarnos en el mismo punto solo que un nivel abajo

        cmp di, 64000d       ; 320 \* 200, límite de la pantalla

        jb LOOP\_I

    pop di

    pop dx

endm

mDibujarEjeX macro

LOCAL LOOP\_I

    push dx

    push di

    xor dl, dl

    mov dl, 1d              ; color de los pixeles del eje X

    mov di, 32000d          ; 320 \* 100

    LOOP\_I:

        xor ax, ax

        mov es:[di], dl

        inc di              ; al incrementar di, nos desplazamos a la derecha

        cmp di, 32320d      ; 320 \* 101

        jb LOOP\_I

    pop di

    pop dx

endm

;Ambos parametros tienen signo (x,y) y color que es el código de colores

mDibujarPixelColor macro x, y, color

    push ax

    push bx

    push di

    push si

    push dx

    FINIT

    xor ax, ax

    xor bx, bx

    xor di, di

    xor si, si

    xor dx, dx

    mov ax, 32159d ; nos posicionamos en el centro Y: 320 \* 100 X: +159

    ; Parte para manejar coordenada: "X"

    mov si, offset numeroEntero1

    mov word ptr[si], ax

    FILD numeroEntero1

    xor ax, ax

    mov si, offset numeroEntero2

    mov ax, word ptr[si]            ; valor de "X" en entero.

    mov si, offset numeroEntero1

    mov word ptr[si], ax

    FILD numeroEntero1

    FADD                       ; desplazamiento en X

    ; Parte para manejar coordenada: "Y"

    xor ax, ax

    mov si, offset numeroEntero3

    mov ax, word ptr[si]            ; valor de "Y" en entero.

    neg ax                     ; se niega para obtener la dirección correcta de subir o bajar en el plano

    mov si, offset numeroEntero1

    mov word ptr[si], ax

    FILD numeroEntero1

    xor ax, ax

    mov ax, 320d

    mov si, offset numeroEntero1

    mov word ptr[si], ax

    FILD numeroEntero1

    FMUL                        ; total distancia en Y

    FADD                        ; desplazamiento en Y

    FISTP numeroEntero1

    mov si, offset numeroEntero1

    mov di, word ptr[si]

    mov dl, color               ; valor de COLOR del pixel

    mov es:[di], dl

    FINIT

    pop dx

    pop si

    pop di

    pop bx

    pop ax

endm

mDibujarPixelColorMejorado macro x, y, color

    push bx

    push ax

    push dx

    push cx

    FINIT

    mov bx, 159d ; nos posicionamos en el centro X: 159

    ; Parte para manejar coordenada: "X"

    mov si, offset numeroEntero1

    mov word ptr[si], bx

    FILD numeroEntero1

    mov word ptr[si], x

    FILD numeroEntero1

    FADD

    FISTP numeroEntero2

    mov bx, 100d ; nos posicionamos en el centro Y: 100

    ; Parte para manejar coordenada: "Y"

    mov si, offset numeroEntero1

    mov word ptr[si], bx

    FILD numeroEntero1

    mov word ptr[si], y

    FILD numeroEntero1

    FADD

    FISTP numeroEntero3

    mov si, offset numeroEntero2

    mov cx, word ptr[si]            ; Coordenada X

    mov si, offset numeroEntero3

    mov dx, word ptr[si]            ; Coordenada Y

    mov ah, 0Ch                     ; Servicio para pintar un pixel

    mov al, color                   ; Color del pixel

    int 10h

    pop cx

    pop dx

    pop ax

    pop bx

endm

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIN DECLARACION DE MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INICIO DECLARACION DE VARIABLES DEL PROGRAMA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.MODEL small ; Sirve para definir atributos del modelo de memoria

.STACK ; Crea el segmento de pila con valor por default de 1KB sino se define

.RADIX 10 ; Declara que el sistema númerico a utilizar será el hexadecimal (16), por default es decimal (10)

.DATA ; Crea el segmento de datos, aquí se declaran variables...

; recordar que el db es 'Define Byte' y define un variable de 8-bit en memoria.

;Variables para el metodo de Newton y Steffense

valorMaximoDeIteraciones dw 5d

tope13 db '$'

coeficienteTolerancia dw 5d

tope2 db '$'

baseDiez dw 10d

tope14 db '$'

gradoTolerancia dw 3d

tope3 db '$'

limiteSuperior dw 1d

tope4 db '$'

limiteInferior dw -1d

tope5 db '$'

valorYFuncion dd ?

tope8 db '$'

valorYFuncionOriginal dd ?

tope11 db '$'

valorYFuncionDerivada dd ?

tope9 db '$'

valorIteracionAnterior dd 0d

tope1 db '$'

valorErrorAbsolutoIteracion dd ?

tope12 db '$'

valorErrorAbsolutoAceptable dd ?

tope7 db '$'

valorDecimalCualquiera dd ?

tope6 db '$'

variableValorDos dw 2d

tope10 db '$'

; Variables para ver lo del resultado en decimal

;variables para imprimir decimales

puntoDecimal db 46d, '$'

signoDecimal db 43d, '$'

banderaFPU dw ?, '$'

variableCero dw ?, '$'

parteDecimal dw ?, '$'

extraerDecimal dw 1000d, '$'

variableValorUno dw 1, '$'

numeroCualquiera dw 8, '$'

otroCualquiera dw 3, '$'

parteEntera dw ?

variable dw ?

; Variables para graficar o dibujar (como se le quiera decir)

valorY       dw ? ; Variable para almacenar el valor de la coordenada Y.

almacenador1 dw ? ; Variable para almacenar

almacenador2 dw ? ; Variable para almacenar

almacenador3 dw ? ; Variable para almacenar

almacenador4 dw ? ; Variable para almacenar

numeroEntero2 dw ?, '$'

numeroEntero3 dw ?, '$'

direccion1 dw ? ; Variable para almacenar direcciones

; Variables para la funcion integral

; array word con salto de 3

coeficiente0Integral db 2 dup(0), 24h ; Posicion 0

coeficiente1Integral db 2 dup(0), 24h ; Posicion 3

coeficiente2Integral db 2 dup(0), 24h ; Posicion 6

coeficiente3Integral db 2 dup(0), 24h ; Posicion 9

coeficiente4Integral db 2 dup(0), 24h ; Posicion 12

coeficiente5Integral db 2 dup(0), 24h ; Posicion 15

numeroEntero1 dw 0, '$'    ; almacena el menos y así.

almacenarContador db 2 dup(0)

salidaNumeros     db 6 dup('$')

cadEntrada db 5 dup(0), 24h

; Variables para la funcion derivada

; array word con salto de 3

coeficiente0Derivada db 2 dup(0), 24h ; Posicion 0

coeficiente1Derivada db 2 dup(0), 24h ; Posicion 3

coeficiente2Derivada db 2 dup(0), 24h ; Posicion 6

coeficiente3Derivada db 2 dup(0), 24h ; Posicion 9

coeficiente4Derivada db 2 dup(0), 24h ; Posicion 12

coeficiente5Derivada db 2 dup(0), 24h ; Posicion 15

; Variables para la funcion original

; array word con salto de 3

coeficiente0 db 2 dup(0), 24h ; Posicion 0

coeficiente1 db 2 dup(0), 24h ; Posicion 3

coeficiente2 db 2 dup(0), 24h ; Posicion 6

coeficiente3 db 2 dup(0), 24h ; Posicion 9

coeficiente4 db 2 dup(0), 24h ; Posicion 12

coeficiente5 db 2 dup(0), 24h ; Posicion 15

signo db 1 dup(0) ; si es 1 es un positivo, si es 0 es un negativo

gradoFuncion db 1 dup(0)

valorBaseNumerica dw 000Ah

preguntaIteracionesMaximos db "Ingresar numero de iteraciones maximo: ", 24h

preguntaCoeficienteTolerancia db "Ingresar el coeficiente de tolerancia: ", 24h

preguntaGradoTolerancia db "Ingresar el grado de tolerancia: ", 24h

preguntarLimiteSuperior db "Ingresar el limite Superior: ", 24h

preguntarLimiteInferior db "Ingresar el limite Inferior: ", 24h

suFuncionEs         db "La funcion generada es: ", 24h

parentesisAbre      db "(", 24h

parentesisCierra    db ")", 24h

signoSuma           db "+", 24h

letraX              db "x^", 24h

preguntaCoeficiente db "Ingrese el coeficiente de x^", 24h

cierrePregunta      db ":", 24h

preguntaGrado db "Ingrese el grado de su funcion: ", 24h

errorDigitoNoValido db "Entrada no valida ", 24h

opcion1           db "Selecciono la opcion 1.", 0Ah, 24h

opcion2           db "Selecciono la opcion 2.", 0Ah, 24h

opcion3           db "Selecciono la opcion 3.", 0Ah, 24h

opcion4           db "Selecciono la opcion 4.", 0Ah, 24h

opcion5           db "Selecciono la opcion 5.", 0Ah, 24h

opcion6           db "Selecciono la opcion 6.", 0Ah, 24h

opcion7           db "Selecciono la opcion 7.", 0Ah, 24h

opcion8           db "Selecciono la opcion 8.", 0Ah, 24h

opcion9           db "Selecciono la opcion 9.", 0Ah, 24h

menu    db "///////////////// MENU /////////////////", 0Ah

                db "(1) Ingresar Funcion.", 0Ah

                db "(2) Imprimir la funcion almacenada.", 0Ah

                db "(3) Imprimir la derivada de la funcion almacenada.", 0Ah

                db "(4) Imprimir la funcion almacenada.", 0Ah

                db "(5) Imprimir la integral de la funcion almacenada.", 0Ah

                db "(6) Graficar la funcion almacenada (original/derivada/integral).", 0Ah

                db "(7) Encontrar los ceros de la funcion por medio del metodo de Newton.", 0Ah

                db "(8) Encontrar los ceros de la funcion por medio del metodo de Steffensen.", 0Ah

                db "(9) Salir de la aplicacion,", 0Ah, 24h

subMenu    db "///////////////// SUBMENU /////////////////", 0Ah

           db "(1) Graficar funcion original.", 0Ah

           db "(2) Graficar funcion derivada.", 0Ah

           db "(3) Graficar funcion integral.", 0Ah, 24h

presioneEnter   db "Presione Enter para continuar...", 0Ah, 24h

errorMenu1      db "Opcion incorrecta, seleccione solo valores (1,2,3,4,5,6,7,8,9).", 0Ah, 24h

saltoLinea      db 0Ah, 24h

adios           db "Hasta la proximaaa!!!!", 24h

.CODE

lInicio:

    ; \*\*\* MAIN \*\*\*\*

    pMain proc

        mov dx, @DATA   ; esto va siempre en el main, inicializar area de datos

        mov ds, dx      ; esto también va siempre en el main

        FINIT ; Inicializando FPU

        call pImprimirMensajeInicial

        jmp lSalir ; terminar proceso si se llega aquí

    pMain endp

    ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROC O ETIQUETAS ESPECIFICOS DE LA APP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    ;---------------------------------------------------------

    pImprimirMensajeInicial proc

    ;

    ; Procedimiento para imprimir el texto inicial

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        lPrint1:    ; imprimir mensaje inicial

            mLimpiarPantalla

            mImprimirCadena menu

            mov ah,01h    ; se carga en la parte alta el servicio 01h, que lee un caracter de la entrada y lo guarda en el registro al.

            int 21h       ; se ejecuta el servicio cargado en ah, ejecuta 01h.

            cmp al, 49    ; Compara si el valor en el registro al es un '3'. al = 0dh, entonces ZF = 1, de lo contrario ZF = 0.

            je  lOpcion1    ; je -> if ZF = 1 then jump. Cerrar programa

            cmp al, 50

            je  lOpcion2

            cmp al, 51

            je  lOpcion3

            cmp al, 52

            je  lOpcion4

            cmp al, 53

            je  lOpcion5

            cmp al, 54

            je  lOpcion6

            cmp al, 55

            je  lOpcion7

            cmp al, 56

            je  lOpcion8

            cmp al, 57

            je  lOpcion9

            jmp lOpcionIncorrecta1

        lOpcion1:

            call pOpcion1

            jmp lPrint1

        lOpcion2:

            call pOpcion2

            jmp lPrint1

        lOpcion3:

            call pOpcion3

            jmp lPrint1

        lOpcion4:

            call pOpcion4

            jmp lPrint1

        lOpcion5:

            call pOpcion5

            jmp lPrint1

        lOpcion6:

            call pOpcion6

            jmp lPrint1

        lOpcion7:

            call pOpcion7

            jmp lPrint1

        lOpcion8:

            call pOpcion8

            jmp lPrint1

        lOpcion9:

            call pOpcion9

            jmp lPrint1

        lOpcionIncorrecta1:

        mImprimirCadena errorDigitoNoValido

        mRepetirSaltoSiNoEs presioneEnter, lPrint1

        ret          ; retorna la direccion la llamada al procedimiento donde se llamo, y la asigna al registro ip, para seguir ejecutando instrucciones después de su llamada.

    pImprimirMensajeInicial endp

    ;---------------------------------------------------------

    pOpcion1 proc

    ;

    ; Procedimiento para la opcion 1 para generar la función

    ; Receives: ---

    ; Returns: coeficientes llenos e imprime la función de una vez en la consola.

    ;---------------------------------------------------------

        xor ax, ax

        mLimpiarPantalla

        mImprimirCadena opcion1

        mImprimirCadena preguntaGrado

        mLeerCaracter ; guarda el caracter en 'AL'

        cmp al, 48

        jb lPrintError2 ; si al es más pequeño que 48

        cmp al, 53

        jg lPrintError2 ; si al es más grande que 53

        jmp guardarGrado

        lPrintError2:

            jmp lPrintError1

        guardarGrado:

            sub al, 48

            mov gradoFuncion, al ; se guarda el valor del grado

            xor ah, ah

            xor cx, cx

            ; limpiar variable almacenarContador

            mov si, offset almacenarContador

            mov word ptr[si], 0000

            ; grado de la función a contador

            mov cx, ax

            ; inicializando coeficientes

            mLimpiarCadena cadEntrada

            mLimpiarVariableByte coeficiente0

            mLimpiarVariableByte coeficiente1

            mLimpiarVariableByte coeficiente2

            mLimpiarVariableByte coeficiente3

            mLimpiarVariableByte coeficiente4

            mLimpiarVariableByte coeficiente5

            mLimpiarVariableByte signo

            mImprimirCadena saltoLinea

            ; di: usando para almacenar la posición en la cadena de entrada

            ; si: puede ser volatil

            ; ax: se usa en varios metodos

            ; bx: volatil

            ; Pedir coeficientes y guardarlos

            lciclo1:

                mLimpiarCadena cadEntrada

                mLimpiarCadena numeroEntero1

                ; guardando registro en almacenar contador

                mov si, offset almacenarContador

                mov word ptr[si], cx

                mImprimirCadena preguntaCoeficiente

                mImprimirValorRegistroByte cl

                mImprimirCadena cierrePregunta

                mLeerCadenaConsola cadEntrada

                ; Extraer el valor de una cadena a un registro

                mov di, offset cadEntrada

                mov al, byte ptr[di]

                ; Preguntar si es un (+) o (-)

                cmp al, 43

                je lEnteroPositivo1

                cmp al, 45

                je lEnteroNegativo1

                mIsDigit lPrintError1 ; retorna ZF = 1 si es un digito, salta a lPrintError 1 si no lo es

                mov bx, 0001

                mov numeroEntero1, bx

                FILD numeroEntero1

                jmp lLeerNumero

                lEnteroPositivo1:

                    mov bx, 0001

                    mov numeroEntero1, bx

                    FILD numeroEntero1

                    inc di ; te desplazas en la cadena

                    jmp lLeerNumero

                lEnteroNegativo1:

                    mov bx, 0001

                    neg bx

                    mov numeroEntero1, bx

                    FILD numeroEntero1

                    inc di ; te desplazas en la cadena

                    jmp lLeerNumero

                lLeerNumero:

                    xor bx, bx

                    xor ax, ax

                    mov al, byte ptr[di]

                    mIsDigit lPrintError1

                    sub al, 30h         ; restamos 48 para que del valor del ascii ahora tengamos el valor aritmético del dígito.

                    add bl, al

                    inc di ; nos desplazamo en la cadena

                    cmp byte ptr[di], 0Dh

                    je lAceptarCoeficiente

                    xor ax, ax

                    mov al, byte ptr[di]

                    mIsDigit lPrintError1

                    sub al, 30h         ; restamos 48 para que del valor del ascii ahora tengamos el valor aritmético del dígito.

                    xchg ax, bx                 ; esto porque en bx se almacena el valor aritmético

                    mul valorBaseNumerica ; Ax = Ax \* 10

                    xchg ax, bx                 ; Después de realizar la mult. devolvemos todo a su lugar

                    add bl, al

                lAceptarCoeficiente:

                    xor ax, ax

                    mov al, bl

                    xor bx, bx

                    mLimpiarCadena numeroEntero1

                    mov si, offset numeroEntero1

                    mov byte ptr[si], al ; (+1) para que me lo guarde así 00 09 y no 09 00

                    FILD numeroEntero1

                    FMUL

                    FISTP numeroEntero1     ; Extraer resultado

                lGuardarCoeficiente:

                    xor ax, ax

                    xor bx, bx

                    mov si, offset almacenarContador

                    mov ax, word ptr[si]

                    cmp al, 5

                    je lGuardarCoeficiente5

                    cmp al, 4

                    je lGuardarCoeficiente4

                    cmp al, 3

                    je lGuardarCoeficiente3

                    cmp al, 2

                    je lGuardarCoeficiente2

                    cmp al, 1

                    je lGuardarCoeficiente1

                    cmp al, 0

                    je lGuardarCoeficiente0

                    lGuardarCoeficiente5:

                        mov si, offset coeficiente5

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                    lGuardarCoeficiente4:

                        mov si, offset coeficiente4

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                    lGuardarCoeficiente3:

                        mov si, offset coeficiente3

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                    lGuardarCoeficiente2:

                        mov si, offset coeficiente2

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                    lGuardarCoeficiente1:

                        mov si, offset coeficiente1

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                    lGuardarCoeficiente0:

                        mov si, offset coeficiente0

                        mov di, offset numeroEntero1

                        mov ax, word ptr[di]

                        mov word ptr[si], ax

                        jmp lContinuarCiclo1

                lContinuarCiclo1:

                mImprimirCadena saltoLinea

                ; extrayendo valor de almacenar contador hacia el registro

                mov si, offset almacenarContador

                mov cx, word ptr[si]

            dec cx

            cmp cx, 0000

            jl lAceptarEntrada

            jmp lciclo1

        lPrintError1:

            mImprimirCadena saltoLinea

            mImprimirCadena errorDigitoNoValido

            mImprimirCadena saltoLinea

            mRepetirSaltoSiNoEs presioneEnter, lSalirOpcion1

        lAceptarEntrada:

            call pGenerarDerivada

            call pGenerarIntegral

        lSalirOpcion1:

        ret

    pOpcion1 endp

    ;---------------------------------------------------------

    pOpcion2 proc

    ;   Procedimiento para la opcion 2, que es imprimir la función

    ;   Receives: array de coeficientes de la función original

    ;   Use:    [Cx] contador para recorrer los terminos de la función de grado 5 y otras cosas.

    ;           [Si] para guardar el contador

    ;   Returns: Impresion en pantalla

    ;---------------------------------------------------------

        mLimpiarPantalla

        mImprimirCadena opcion2

        mImprimirCadena saltoLinea

        ;Imprimir funcion

        mov dx, offset coeficiente0

        call pImprimirFuncionPolinomica

        mImprimirCadena saltoLinea

        mRepetirSaltoSiNoEs presioneEnter, lSalirOpcion2

        lSalirOpcion2:

        ret

    pOpcion2 endp

    ;---------------------------------------------------------

    pOpcion3 proc

    ;

    ; Procedimiento para la opcion 3, opcion para realizar la derivada de la función e imprimirla

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mImprimirCadena opcion3

        mImprimirCadena saltoLinea

        ;Imprimir funcion

        mov dx, offset coeficiente0Derivada

        call pImprimirFuncionPolinomica

        mImprimirCadena saltoLinea

        mRepetirSaltoSiNoEs presioneEnter, lSalirOpcion3

        lSalirOpcion3:

        ret

    pOpcion3 endp

    ;---------------------------------------------------------

    pOpcion4 proc

    ;

    ; Procedimiento para la opcion 4, imprimir la integral

    ; Receives: todos los registros.

    ; Returns: Salida en consola de integral.

    ;---------------------------------------------------------

        mLimpiarPantalla

        mImprimirCadena opcion4

        mImprimirCadena saltoLinea

        ;Imprimir funcion

        mov dx, offset coeficiente0Integral

        call pImprimirFuncionPolinomica

        mImprimirCadena saltoLinea

        mRepetirSaltoSiNoEs presioneEnter, lSalirOpcion4

        lSalirOpcion4:

        ret

    pOpcion4 endp

    ;---------------------------------------------------------

    pOpcion5 proc

    ;

    ; Procedimiento para la opcion 1

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mImprimirCadena opcion5

        ret

    pOpcion5 endp

    ;---------------------------------------------------------

    pOpcion6 proc

    ;

    ; Procedimiento para la opcion 1

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mImprimirCadena opcion6

        mImprimirCadena subMenu

            mov ah,01h    ; se carga en la parte alta el servicio 01h, que lee un caracter de la entrada y lo guarda en el registro al.

            int 21h       ; se ejecuta el servicio cargado en ah, ejecuta 01h.

            cmp al, 49    ; Compara si el valor en el registro al es un '3'. al = 0dh, entonces ZF = 1, de lo contrario ZF = 0.

            je  lOpcion61    ; je -> if ZF = 1 then jump. Cerrar programa

            cmp al, 50

            je  lOpcion62

            cmp al, 51

            je  lOpcion63

            jmp lOpcionIncorrecta2

            lOpcion61:

            call pOpcion61

            jmp lSalirOpcion6

            lOpcion62:

            call pOpcion62

            jmp lSalirOpcion6

            lOpcion63:

            call pOpcion63

            jmp lSalirOpcion6

            lOpcionIncorrecta2:

            mImprimirCadena errorDigitoNoValido

            mRepetirSaltoSiNoEs presioneEnter, lSalirOpcion6

            lSalirOpcion6:

        ret

    pOpcion6 endp

    ;---------------------------------------------------------

    pOpcion61 proc

    ;

    ; Procedimiento para la opcion 6.1 imprimir la función original

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mActivarModoVideo

        mDibujarEjeX

        mDibujarEjeY

        mov di, offset coeficiente0

        call pDibujarGrafica

        REPETIR:

        CALL TECLA

        CMP AL,27                       ; Tecla ESC

        JE SALIR

        XOR CX, CX

        XOR DX, DX

        MOV CL, AL

        MOV DL, AH

        JMP REPETIR

        SALIR:

        mDesactivarModoVideo

        ret

    pOpcion61 endp

    ;---------------------------------------------------------

    pOpcion62 proc

    ;

    ; Procedimiento para la opcion 6.2 imprimir la derivada

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mActivarModoVideo

        mDibujarEjeX

        mDibujarEjeY

        mov di, offset coeficiente0Derivada

        call pDibujarGrafica

        REPETIR2:

        CALL TECLA

        CMP AL,27                       ; Tecla ESC

        JE SALIR2

        XOR CX, CX

        XOR DX, DX

        MOV CL, AL

        MOV DL, AH

        JMP REPETIR2

        SALIR2:

        mDesactivarModoVideo

        ret

    pOpcion62 endp

    ;---------------------------------------------------------

    pOpcion63 proc

    ;

    ; Procedimiento para la opcion 6.3 imprimir una integral

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mActivarModoVideo

        mDibujarEjeX

        mDibujarEjeY

        mov di, offset coeficiente0Integral

        call pDibujarGrafica

        REPETIR3:

        CALL TECLA

        CMP AL,27                       ; Tecla ESC

        JE SALIR3

        XOR CX, CX

        XOR DX, DX

        MOV CL, AL

        MOV DL, AH

        JMP REPETIR3

        SALIR3:

        mDesactivarModoVideo

        ret

    pOpcion63 endp

    ;---------------------------------------------------------

    pOpcion7 proc

    ;

    ; Procedimiento para la opcion 7, encontrar los ceros por medio del método de Newton

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mImprimirCadena opcion7

        mImprimirCadena preguntaIteracionesMaximos

        ret

    pOpcion7 endp

    ;---------------------------------------------------------

    pOpcion8 proc

    ;

    ; Procedimiento para la opcion 1

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mImprimirCadena opcion8

        ret

    pOpcion8 endp

    ;---------------------------------------------------------

    pOpcion9 proc

    ;

    ; Procedimiento para la opcion 1

    ; Receives: ---

    ; Returns: ---

    ;---------------------------------------------------------

        mLimpiarPantalla

        mImprimirCadena opcion9

        mRepetirSaltoSiNoEs presioneEnter, lSalir

        ret

    pOpcion9 endp

    pLeerNumeroEnteroEntrada proc

    ;

    ; Procedimiento para recibir un numero entero en la entrada

    ; Receives: [si] offset de la variable donde se va a guardar el resultado

    ; Returns: ---

    ;---------------------------------------------------------

        push si

        mLeerCadenaConsola cadEntrada

        ; Extraer el valor de una cadena a un registro

        mov di, offset cadEntrada

        mov al, byte ptr[di]

        ; Preguntar si es un (+) o (-)

        cmp al, 43

        je lEnteroPositivo3

        cmp al, 45

        je lEnteroNegativo3

        mIsDigit lPrintError3 ; retorna ZF = 1 si es un digito, salta a lPrintError 1 si no lo es

        mov bx, 0001

        mov numeroEntero1, bx

        FILD numeroEntero1

        jmp lLeerNumero3

        lEnteroPositivo3:

            mov bx, 0001

            mov numeroEntero1, bx

            FILD numeroEntero1

            inc di ; te desplazas en la cadena

            jmp lLeerNumero3

        lEnteroNegativo3:

            mov bx, 0001

            neg bx

            mov numeroEntero1, bx

            FILD numeroEntero1

            inc di ; te desplazas en la cadena

            jmp lLeerNumero3

        lLeerNumero3:

            xor bx, bx

            xor ax, ax

            mov al, byte ptr[di]

            mIsDigit lPrintError3

            sub al, 30h         ; restamos 48 para que del valor del ascii ahora tengamos el valor aritmético del dígito.

            add bl, al

            inc di ; nos desplazamo en la cadena

            cmp byte ptr[di], 0Dh

            je lAceptarCoeficiente3

            xor ax, ax

            mov al, byte ptr[di]

            mIsDigit lPrintError3

            sub al, 30h         ; restamos 48 para que del valor del ascii ahora tengamos el valor aritmético del dígito.

            xchg ax, bx                 ; esto porque en bx se almacena el valor aritmético

            mul valorBaseNumerica ; Ax = Ax \* 10

            xchg ax, bx                 ; Después de realizar la mult. devolvemos todo a su lugar

            add bl, al

        lAceptarCoeficiente3:

            xor ax, ax

            mov al, bl

            xor bx, bx

            mLimpiarCadena numeroEntero1

            mov si, offset numeroEntero1

            mov byte ptr[si], al ; (+1) para que me lo guarde así 00 09 y no 09 00

            FILD numeroEntero1

            FMUL

            FISTP numeroEntero1     ; Extraer resultado

            lPrintError3:

        ret

    pLeerNumeroEnteroEntrada endp

    ;---------------------------------------------------------

    pImprimirFuncionPolinomica proc

    ;

    ; Procedimiento para imprimir una función polinomica.

    ; Hago esto en un procedimiento porque llamar macros con otras macros me realentiza a mi, mi DOSBox.

    ; Receives: [dx] offset del array de coeficientes que tiene que ser de longitud 6

    ; Use:      una variable para almacenar la dirección del offset al principio

    ; Returns: ---

    ;---------------------------------------------------------

        ; Guardando la dirección de la variable parametro

        mov si, offset direccion1

        mov word ptr[si], dx

        xor dx, dx

        mImprimirCadena saltoLinea

        ;Imprimir funcion

        ;Imprimir x^Cx

        mov cx, 0005

            lImprimirTermino5:

                ; guardando registro en almacenar contador

                mov si, offset almacenarContador

                mov word ptr[si], cx

                ; Calculando la direccion del valor del array según el número de iteración

                mov ax, cx

                mov bx, 0003

                mul bx

                ; Extrayendo direccion de la variable parametro de memoria y enviandola [Si]

                mov di, offset direccion1

                mov si, word ptr[di]

                add si, ax

                cmp word ptr[si], 0000

                je lContinuarCiclo5

                mLimpiarCadenaEntero salidaNumeros ; limpio la variable por si tiene basura

                ; Calculando la direccion del valor del array según el número de iteración

                mov ax, cx

                mov bx, 0003

                mul bx

                ; Extrayendo direccion de la variable parametro de memoria y enviandola [Si]

                mov di, offset direccion1

                mov si, word ptr[di]

                add si, ax

                mIntToString salidaNumeros, si

                mImprimirCadena salidaNumeros

                ; extrayendo valor de almacenar contador hacia el registro

                mov si, offset almacenarContador

                mov cx, word ptr[si]

                mImprimirCadena letraX

                mImprimirValorRegistroByte cl

                lContinuarCiclo5:

                ; extrayendo valor de almacenar contador hacia el registro

                mov si, offset almacenarContador

                mov cx, word ptr[si]

            dec cx

            cmp cx, 0000

            jl lFuncionImpresa5

            jmp lImprimirTermino5

        lFuncionImpresa5:

        ret

    pImprimirFuncionPolinomica endp

    ;---------------------------------------------------------

    pGenerarDerivada proc

    ;

    ; Procedimiento para generar la derivida de la funcion original

    ; Receives: ---

    ; Returns: cambios en el array de los coeficientes de la derivada.

    ;---------------------------------------------------------

        ; Parte de generar derivada

        FINIT ; Inicializando FPU

        ; inicializando coeficientes

        mLimpiarCadena cadEntrada

        mLimpiarVariableByte coeficiente0Derivada

        mLimpiarVariableByte coeficiente1Derivada

        mLimpiarVariableByte coeficiente2Derivada

        mLimpiarVariableByte coeficiente3Derivada

        mLimpiarVariableByte coeficiente4Derivada

        mLimpiarVariableByte coeficiente5Derivada

        mLimpiarVariableByte signo

        ;Generar Derivada

        ;Aplicar derivada a cada x^Cx

        mov cx, 0005

        lGenerarDerivada:

            FINIT

            ; guardando registro en almacenar contador

            mov si, offset almacenarContador

            mov word ptr[si], cx

            ; Calculando la direccion del valor del array según el número de iteración [si]

            mov ax, cx

            mov bx, 0003

            mul bx

            mov si, offset coeficiente0

            add si, ax

            cmp word ptr[si], 0000

            je lContinuarCiclo2

            ; Calculando la direccion del valor del array según el número de iteración [di]

            mov ax, cx

            mov bx, 0003

            mul bx

            mov di, offset coeficiente0Derivada

            add di, ax

            sub di, 0003 ; pero al ser la derivada debe ser el coeficiente menor a el coeficiente original

            ; se tiene en [si] el coeficiente original y en el [di] el coeficiente derivada

            mLimpiarCadena numeroEntero1

            xor dx, dx

            mov dx, word ptr[si]

            mov numeroEntero1, dx

            FILD numeroEntero1 ; ingresa el número al FPU

            mLimpiarCadena numeroEntero1

            mov numeroEntero1, cx

            FILD numeroEntero1 ; ingresa el número al FPU

            ; realizar operaciones entre los números mediante el FPU

            FMUL

            FISTP numeroEntero1 ; realiza la múltiplicación y la extrae del FPU y la guarda en numeroEntero

            xor dx, dx

            mov dx, numeroEntero1

            mov word ptr[di], dx

            lContinuarCiclo2:

            ; extrayendo valor de almacenar contador hacia el registro

            mov si, offset almacenarContador

            mov cx, word ptr[si]

        dec cx

        cmp cx, 0000

        je lSalirProcDerivada

        jmp lGenerarDerivada

        lSalirProcDerivada:

        ret

    pGenerarDerivada endp

    ;---------------------------------------------------------

    pGenerarIntegral proc

    ;

    ; Procedimiento para generar la integral de la funcion original

    ; Receives: ---

    ; Returns: cambios en el array de los coeficientes de la integral.

    ;---------------------------------------------------------

        FINIT ; Inicializando FPU

        ; inicializando coeficientes

        ; inicializando coeficientes

        mLimpiarCadena cadEntrada

        mLimpiarVariableByte coeficiente0Integral

        mLimpiarVariableByte coeficiente1Integral

        mLimpiarVariableByte coeficiente2Integral

        mLimpiarVariableByte coeficiente3Integral

        mLimpiarVariableByte coeficiente4Integral

        mLimpiarVariableByte coeficiente5Integral

        mLimpiarVariableByte signo

        ;Generar Integral

        ;Aplicar Integral a cada x^Cx

        mov cx, 0004

        lGenerarIntegral:

            FINIT

            ; guardando registro en almacenar contador

            mov si, offset almacenarContador

            mov word ptr[si], cx

            ; Calculando la direccion del valor del array según el número de iteración [si]

            mov ax, cx

            mov bx, 0003

            mul bx

            mov si, offset coeficiente0

            add si, ax

            cmp word ptr[si], 0000

            je lContinuarCiclo3

            ; Calculando la direccion del valor del array según el número de iteración [di]

            mov ax, cx

            mov bx, 0003

            mul bx

            mov di, offset coeficiente0Integral

            add di, ax

            add di, 0003 ; pero al ser la Integral debe ser el coeficiente mayor a el coeficiente original

            ; se tiene en [si] el coeficiente original y en el [di] el coeficiente Integral

            mLimpiarCadena numeroEntero1

            xor dx, dx

            mov dx, word ptr[si]

            mov numeroEntero1, dx

            FILD numeroEntero1                          ; ingresa el número al FPU

            mLimpiarCadena numeroEntero1

            inc cx                                      ; esto porque la integral es coefieciente / (grado Actual + 1)

            mov numeroEntero1, cx

            dec cx                                      ; esto es para que siga el ciclo normal

            FILD numeroEntero1                          ; ingresa el número al FPU

            ; realizar operaciones entre los números mediante el FPU

            FDIV

            FISTP numeroEntero1 ; realiza la múltiplicación y la extrae del FPU y la guarda en numeroEntero

            xor dx, dx

            mov dx, numeroEntero1

            mov word ptr[di], dx

            lContinuarCiclo3:

            ; extrayendo valor de almacenar contador hacia el registro

            mov si, offset almacenarContador

            mov cx, word ptr[si]

        dec cx

        cmp cx, 0000

        jl lSalirProcIntegral

        jmp lGenerarIntegral

        lSalirProcIntegral:

        ret

    pGenerarIntegral endp

    ;---------------------------------------------------------

    pDibujarGrafica PROC

    ; Todas las funciones van de X: -10 a X: +10

    ; Procedimiento para dibujar una funcion polinomica

    ; Receives: [di] direccion de la posicion 0 del array de coeficientes

    ; Returns: Dibujo en pantalla.

    ;---------------------------------------------------------

        mov si, offset almacenador1

        mov word ptr[si], di

        xor si, si

        xor di, di

        ;Imprimir funcion en el intervalo de X: (-100, +100)

        mov cx, -10d

        lDibujarPuntoGrafica:

            FINIT

            ; guardando registro en almacenar contador

            mov si, offset almacenador2

            mov word ptr[si], cx

            ;Calcular cada x^Cx y sumarlos, para determinar la coordenada "Y"

            mov cx, 0005

            lCalcularCoordenadas:

                ; guardando registro en almacenar contador

                mov si, offset almacenador3

                mov word ptr[si], cx

                ; Calculando la direccion del valor del array según el número de iteración

                mov ax, cx

                mov bx, 0003

                mul bx

                ; Extrayendo direccion de la variable parametro de memoria y enviandola [Si]

                mov di, offset almacenador1

                mov si, word ptr[di]

                add si, ax          ; Se obtiene la posicion del array acuerdo a la iteracion del ciclo

                cmp word ptr[si], 0000 ; Si el coeficiente vale 0, saltarselo así se ahorra tiempo

                je lContinucarCalculoCoordenadas

                ; Calcular valor de C\*x^n

                ; extrayendo valor de almacenar contador hacia el registro (n)

                mov di, offset almacenador3

                mov cx, word ptr[di]

                ; guardando la base (x)

                FILD almacenador2

                ; realizar parte de potencia x^n

                lCicloPotencia:

                cmp cx, 1d

                je lSalirCicloPotencia

                cmp cx, 0d

                je lSalirPotenciaCasoCero

                    FIMUL almacenador2

                dec cx

                jmp lCicloPotencia

                lSalirPotenciaCasoCero:

                FINIT

                mov di, offset numeroEntero1

                mov ax, 1d

                mov word ptr[di], ax

                FILD numeroEntero1

                lSalirCicloPotencia:

                ; realizar multiplicación de C \* [x^n]

                mov ax, word ptr[si]

                mov di, offset numeroEntero1

                mov word ptr[di], ax

                FILD numeroEntero1

                FMUL        ; ST(0) = resultado de multiplicación de C \* [x^n]

                FILD valorY ; ahora quedaría así el FPU, ST(0) = valorY, ST(1) = C\*[x^n]+C[x^n-1]+...C[x^0]

                FADD

                FISTP valorY ; guardamos el resultado en el valor Y nuevamente.

                lContinucarCalculoCoordenadas:

                ; extrayendo valor de almacenar contador hacia el registro

                mov si, offset almacenador3

                mov cx, word ptr[si]

                dec cx

                cmp cx, 0000

                jl lContinuarDibujarPuntoGrafica

                jmp lCalcularCoordenadas

            lContinuarDibujarPuntoGrafica:

            ; extrayendo valor de almacenar contador hacia el registro

            mov si, offset almacenador2

            mov cx, word ptr[si]

            ; Dibujar punto en la grafica con las coordenadas

            mov si, offset valorY

            mov ax, word ptr[si]

            ; Cx = X y Ax = Y

            ; Antes de pintar validar que las coordenadas no se salgan de la pantalla, porque sino loquea

            cmp ax, 100d

            jge lContinuarCSP1

            cmp ax, -100d

            jge lSiPintar

            lContinuarCSP1:

            jmp lContinuarConSiguientePunto

            lSiPintar:

            ; Cx = X y Ax = Y

            mov dx, ax ; y -> dx = ax

            neg dx     ; se niega para que suba de acuerdo a la coordenada la grafica

            mDibujarPixelColorMejorado cx, dx, 63h

            lContinuarConSiguientePunto:

            ; Reiniciar coordena Y

            mov si, offset valorY

            mov word ptr[si], 0000h

            inc cx

            cmp cx, 11d

            je lTerminarDibujo

            jmp lDibujarPuntoGrafica

        lTerminarDibujo:

        ret

    pDibujarGrafica ENDP

    ; ------------------- Obtiene respuesta del teclado --------------------

    TECLA PROC

        MOV AH, 10H         ; Petici�n entrada del teclado al BIOS

        INT 16H             ; Llama al BIOS

        RET

    TECLA ENDP

;------------------------

; etiqueta utilizada para cerrar el programa

;------------------------

lSalir:

    ; Cuando se termina el programa siempre hay que mandar esto o .exit que es lo mismo

    mImprimirCadena adios

    mov al, 16h  ; retorno funcion main

    mov ah, 04Ch ; se carga en la parte alta el servicio 04Ch, devuelve el control al sistema, termina proceso

    int 21h ; se ejecuta el servicio cargado en ah, ejecuta 04Ch.

    ; Recordar que la interrupcion 21h se utiliza para entradas y salidas, files, administracion de memoria y llamadas a funciones.

END lInicio