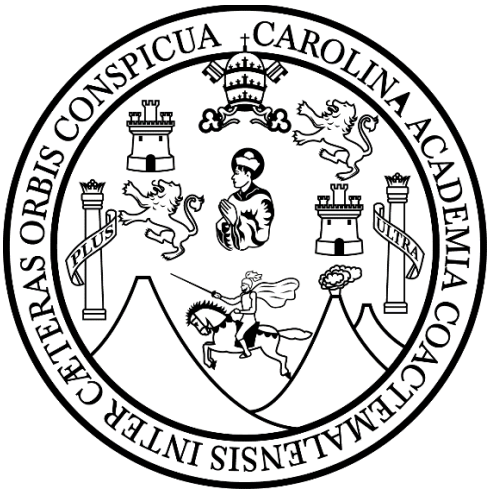
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1



MANUAL TÉCNICO

PROYECTO 2 FASE 1

MARVIN ALEXIS ESTRADA FLORIAN
201800476

INTRODUCCIÓN

El lenguaje ensamblador representa un lenguaje de programación más cercano al lenguaje máquina que suelen comprender las computadoras, es por esta razón que es muy importante conocer el uso de este tipo de lenguaje, que a su vez permite comprender de mejor manera la forma en que el CPU entiende, interpreta y ejecuta las distintas instrucciones de códigos de programación de alto nivel. El proyecto consiste en la creación de una calculadora gráfica, utilizando como herramienta un ensamblador de x86, el coprocesador matemático e interrupciones de DOS, dando como resultado en esta primera fase, mostrar el resultado al ingresar una función desde grado 1 hasta el 5 como su derivada e integral respectiva

CODIGO

Comenzando se tienen las siguientes macros “printString” y “printChar”, las cuales ayudan a optimizar la impresión en consola de cadenas y caracteres respectivamente, utilizando la interrupción 21h.

```
printString macro singleString
    mov ah, 09h
    lea dx, singleString
    int 21h
endm

printChar macro singleChar
    mov ah, 02h
    mov dl, singleChar
    int 21h
endm
```

Posteriormente se tiene el codigo del menú principal, donde se imprimen las diversas opciones del anterior mencionado, validando que esta entrada sea valida, como también limpiar la consola con el procedimiento “cleanScr”, el cual se explicará posteriormente.

```
        mov ax, seg @data;Cargar datos al data segment
        mov ds, ax
        menu:;Mostrar el menu
            call cleanScr
            printString msg0
            printString msg1
            printString msg0
            printString msg6
            printString msg7
            printString msg8
            printString msg9
            printString msg10
            printString msg11
            printString msg12
            printString msg13
            printString msg0
            printString msg14
```

```
mov ah, 01h;Solicitar la opcion del menu
int 21h

;Comparaciones para saber que opcion se ingreso
cmp al, 31h
je auxOpcion1
cmp al, 32h
je auxOpcion2
cmp al, 33h
je auxOpcion3
cmp al, 34h
je auxOpcion4
cmp al, 38h
je auxOpcion8

printChar 10d;Opcion fuera de rango
printString msg23
mov ah, 01h
int 21h
jmp menu
```

Las siguientes etiquetas se utilizan debido a que el código con estos saltos estaban fuera de rango por la longitud de este, por lo que se utilizan saltos y etiquetas auxiliares para lograr alcanzar estos destinos, las etiquetas nombradas como “opcionN”, nos llevan a las opciones solicitadas en el menú para esta fase del proyecto:

```
auxOpcion1:
    jmp opcion1
auxOpcion2:
    jmp opcion2
auxOpcion3:
    jmp opcion3
auxOpcion4:
    jmp opcion4
auxOpcion8:
    jmp opcion8
```

La siguiente porción de código nos lleva a ejecutar la primera opción del menú principal, donde se limpia la pantalla con el procedimiento “cleanScr”, posteriormente se solicita el grado de la ecuación al usuario, según el grado ingresado se va a un salto especifico donde se le solicita al usuario ingresar los coeficientes y signos respectivos de la función a almacenar.

```
opcion1:
            call cleanScr
            printChar 10d
            printString msg0
            printString msg2
            printString msg0
            printString msg15
            mov ah, 01h;Solicitar grado
            int 21h

            mov grade, al
            cmp al, 31h
            je auxOpcion1_1;Ingresar ecuacion grado 1
            cmp al, 32h
            je auxOpcion1_2;Ingresar ecuacion grado 2
            cmp al, 33h
            je auxOpcion1_3;Ingresar ecuacion grado 3
            cmp al, 34h
            je auxOpcion1_4;Ingresar ecuacion grado 4
            cmp al, 35h
            je auxOpcion1_5;Ingresar ecuacion grado 5

            printChar 10d;Opcion fuera de rango
            printString msg26
            mov ah, 01h
            int 21h
            call restartVariables
            jmp menu


            auxOpcion1_1:
                jmp opcion1_1
            auxOpcion1_2:
                jmp opcion1_2
            auxOpcion1_3:
                jmp opcion1_3
            auxOpcion1_4:
                jmp opcion1_4
            auxOpcion1_5:
```

```
                jmp opcion1_5
```

Dando este salto como ejemplo, se solicita inicialmente el signo del coeficiente, para luego solicitar el coeficiente de 2 cifras, donde si se ingresa un carácter no valido, esto lleva a mostrarle el error al usuario, si todo es correcto se muestra un mensaje que la función ha sido almacenada.

```
opcion1_5:;Grado 5
                cmp grade, 35h
                jne notwriteText5
                printChar 10d
                printString msg16
                notwriteText5:

                call signProof;Solicitar signo 5
                jne fail5
                mov sg5, al
                printChar 20h

                call numberProof;Solicitar numero mayor 5
                jne fail5
                add al, 30h
                mov nu5_1, al

                call numberProof;Solicitar numero menor 5
                jne fail5
                add al, 30h
                mov nu5_0, al

                printChar 58h
                printChar 5eh
                printChar 35h
                printChar 20h
                jmp opcion1_4

                fail5:;Error char ingresado incorrecto
                    call restartVariables
                    printChar 10d
                    printString msg25
                    mov ah, 01h
                    int 21h
                    jmp menu
```

Esta es la segunda opción del menú principal donde se imprime la función almacenada, utilizando la misma forma de impresión de cuando el usuario ingresa la función, haciendo saltos dependiendo del grado almacenado e imprimiendo el respectivo contenido de este grado.

```
opcion2:
            ;Comparacion para saber si ya ingreso una funcion
            cmp grade, 00h
            je auxFail2_2
            jmp notFail2_2

            auxFail2_2:
                jmp fail2_2
            notFail2_2:

            printChar 10d
            printString msg17

            cmp grade, 35h
            je auxWrite5
            cmp grade, 34h
            je auxWrite4
            cmp grade, 33h
            je auxWrite3
            cmp grade, 32h
            je auxWrite2
            cmp grade, 31h
            je auxWrite1

            auxWrite5:
                jmp write5
            auxWrite4:
                jmp write4
            auxWrite3:
                jmp write3
            auxWrite2:
                jmp write2
            auxWrite1:
                jmp write1

          write5:;Si es de grado 5 se imprime lo almacenado en este apartado
            printChar sg5
            printChar 20h
            printChar nu5_1
            printChar nu5_0
```

```
            printChar 58h
            printChar 5eh
            printChar 35h
```

Esta es la tercera opción del menú principal donde se imprime la derivada de la función almacenada, utilizando la misma forma de impresión de cuando el usuario ingresa la función, haciendo saltos dependiendo del grado almacenado e imprimiendo el respectivo contenido de este grado, pero en este apartado se realizan las operaciones respectivas de lo que se define como una derivada, multiplicando el exponente o grado por los coeficientes, las constantes se vuelven cero, estas operaciones utilizando los Mnemonicos, **sub**, **add**, **mul**, **aad** y **aam**.

```
opcion3:
            ;Comparacion para saber si ya ingreso una funcion
            cmp grade, 00h
            je auxFail3_3
            jmp notFail3_3

            auxFail3_3:
                jmp fail3_3
            notFail3_3:

            printChar 10d
            printString msg18

            cmp grade, 35h
            je auxDWrite5
            cmp grade, 34h
            je auxDWrite4
            cmp grade, 33h
            je auxDWrite3
            cmp grade, 32h
            je auxDWrite2
            cmp grade, 31h
            je auxDWrite1

            auxDWrite5:
                jmp dWrite5
            auxDWrite4:
                jmp dWrite4
            auxDWrite3:
                jmp dWrite3
            auxDWrite2:
```

```
    jmp dWrite2
auxDWrite1:
    jmp dWrite1

dWrite5:
printChar sg5
printChar 20h
mov al, nu5_1;Se mueve el valor mas significativo
sub al, 30h
mov ch, al

mov al, nu5_0;Se mueve el valor menos significativo
sub al, 30h
mov cl, al

mov ax, cx
aad;Se convierten a hexadecimales para operar

mov bl, 05d
mul bl;Se multiplica por el exponente 5

aam;Retorna la multiplicacion con valores asignados en ax
mov cx, ax
mov rnu5_1, ch
mov rnu5_0, cl
add rnu5_1, 30h
add rnu5_0, 30h
printChar rnu5_1
printChar rnu5_0
printChar 58h
printChar 5eh
printChar 34h

dWrite4:
printChar 20h
printChar sg4
printChar 20h
mov al, nu4_1;Se mueve el valor mas significativo
sub al, 30h
mov ch, al

mov al, nu4_0;Se mueve el valor menos significativo
sub al, 30h
mov cl, al
```

```
        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 04d
        mul bl;Se multiplica por el exponente 4

        aam;Retorna la multiplicacion con valores asignados en ax
        mov cx, ax
        mov rnu4_1, ch
        mov rnu4_0, cl
        add rnu4_1, 30h
        add rnu4_0, 30h
        printChar rnu4_1
        printChar rnu4_0
        printChar 58h
        printChar 5eh
        printChar 33h

        dWrite3:
        printChar 20h
        printChar sg3
        printChar 20h
        mov al, nu3_1;Se mueve el valor mas significativo
        sub al, 30h
        mov ch, al

        mov al, nu3_0;Se mueve el valor menos significativo
        sub al, 30h
        mov cl, al

        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 03d
        mul bl;Se multiplica por el exponente 3

        aam;Retorna la multiplicacion con valores asignados en ax
        mov cx, ax
        mov rnu3_1, ch
        mov rnu3_0, cl
        add rnu3_1, 30h
        add rnu3_0, 30h
        printChar rnu3_1
        printChar rnu3_0
        printChar 58h
```

```
        printChar 5eh
        printChar 32h

        dWrite2:
        printChar 20h
        printChar sg2
        printChar 20h
        mov al, nu2_1;Se mueve el valor mas significativo
        sub al, 30h
        mov ch, al

        mov al, nu2_0;Se mueve el valor menos significativo
        sub al, 30h
        mov cl, al

        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 02d
        mul bl;Se multiplica por el exponente 2

        aam;Retorna la multiplicacion con valores asignados en ax
        mov cx, ax
        mov rnu2_1, ch
        mov rnu2_0, cl
        add rnu2_1, 30h
        add rnu2_0, 30h
        printChar rnu2_1
        printChar rnu2_0
        printChar 58h
        printChar 5eh
        printChar 31h

        dWrite1:
        printChar 20h
        printChar sg1
        printChar 20h
        mov al, nu1_1;Valores se multiplican por 1, solo se transfieren
        mov rnu1_1, al

        mov al, nu1_0
        mov rnu1_0, al
        printChar rnu1_1
        printChar rnu1_0
        printChar 10d
```

```
            printString msg21
            mov ah, 01h
            int 21h
            jmp menu

            fail3_3:
                printChar 10d
                printString msg24
                mov ah, 01h
                int 21h
                jmp menu
```

Esta es la cuarta opción del menú principal donde se imprime la derivada de la función almacenada, utilizando la misma forma de impresión de cuando el usuario ingresa la función, haciendo saltos dependiendo del grado almacenado e imprimiendo el respectivo contenido de este grado, pero en este apartado se realizan las operaciones respectivas de lo que se define como una integral, sumando al respectivo exponente uno, y tomar este y dividirlo entre el coeficiente, como también agregar la respectiva constante + C, estas operaciones utilizando los Mnemonicos, **sub**, **add**, **div**, **aad** y **aam**.

```
opcion4:
            ;Comparacion para saber si ya ingreso una funcion
            cmp grade, 00h
            je auxFail4_4
            jmp notFail4_4

            auxFail4_4:
                jmp fail4_4
            notFail4_4:

            printChar 10d
            printString msg19

            cmp grade, 35h
            je auxIWrite5
            cmp grade, 34h
            je auxIWrite4
            cmp grade, 33h
            je auxIWrite3
            cmp grade, 32h
            je auxIWrite2
```

```
        cmp grade, 31h
        je auxIWrite1

        auxIWrite5:
            jmp iWrite5
        auxIWrite4:
            jmp iWrite4
        auxIWrite3:
            jmp iWrite3
        auxIWrite2:
            jmp iWrite2
        auxIWrite1:
            jmp iWrite1

        iWrite5:
        printChar sg5
        printChar 20h
        mov al, nu5_1;Se mueve el valor mas significativo
        sub al, 30h
        mov ch, al

        mov al, nu5_0;Se mueve el valor menos significativo
        sub al, 30h
        mov cl, al

        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 06d
        div bl;Se divide por el exponente 6

        aam;Retorna la division con valores asignados en ax
        mov cx, ax
        mov rnu5_1, ch
        mov rnu5_0, cl
        add rnu5_1, 30h
        add rnu5_0, 30h
        printChar rnu5_1
        printChar rnu5_0
        printChar 58h
        printChar 5eh
        printChar 36h

        iWrite4:
        printChar 20h
```

```
        printChar sg4
        printChar 20h
        mov al, nu4_1;Se mueve el valor mas significativo
        sub al, 30h
        mov ch, al

        mov al, nu4_0;Se mueve el valor menos significativo
        sub al, 30h
        mov cl, al

        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 05d
        div bl;Se divide por el exponente 5

        aam;Retorna la division con valores asignados en ax
        mov cx, ax
        mov rnu4_1, ch
        mov rnu4_0, cl
        add rnu4_1, 30h
        add rnu4_0, 30h
        printChar rnu4_1
        printChar rnu4_0
        printChar 58h
        printChar 5eh
        printChar 35h

        iWrite3:
        printChar 20h
        printChar sg3
        printChar 20h
        mov al, nu3_1;Se mueve el valor mas significativo
        sub al, 30h
        mov ch, al

        mov al, nu3_0;Se mueve el valor menos significativo
        sub al, 30h
        mov cl, al

        mov ax, cx
        aad;Se convierten a hexadecimales para operar

        mov bl, 04d
        div bl;Se divide por el exponente 4
```

```
aam;Retorna la division con valores asignados en ax
mov cx, ax
mov rnu3_1, ch
mov rnu3_0, cl
add rnu3_1, 30h
add rnu3_0, 30h
printChar rnu3_1
printChar rnu3_0
printChar 58h
printChar 5eh
printChar 34h

iWrite2:
printChar 20h
printChar sg2
printChar 20h
mov al, nu2_1;Se mueve el valor mas significativo
sub al, 30h
mov ch, al

mov al, nu2_0;Se mueve el valor menos significativo
sub al, 30h
mov cl, al

mov ax, cx
aad;Se convierten a hexadecimales para operar

mov bl, 03d
div bl;Se divide por el exponente 3

aam;Retorna la division con valores asignados en ax
mov cx, ax
mov rnu2_1, ch
mov rnu2_0, cl
add rnu2_1, 30h
add rnu2_0, 30h
printChar rnu2_1
printChar rnu2_0
printChar 58h
printChar 5eh
printChar 33h

iWrite1:
printChar 20h
```

```
printChar sg1
printChar 20h
mov al, nu1_1;Se mueve el valor mas significativo
sub al, 30h
mov ch, al

mov al, nu1_0;Se mueve el valor menos significativo
sub al, 30h
mov cl, al

mov ax, cx
aad;Se convierten a hexadecimales para operar

mov bl, 02d
div bl;Se divide por el exponente 2

aam;Retorna la division con valores asignados en ax
mov cx, ax
mov rnu1_1, ch
mov rnu1_0, cl
add rnu1_1, 30h
add rnu1_0, 30h
printChar rnu1_1
printChar rnu1_0
printChar 58h
printChar 5eh
printChar 32h

printChar 20h
printChar sg0
printChar 20h
mov al, nu0_1;Valores se dividen por 1, solo se transfieren
mov rnu0_1, al

mov al, nu0_0
mov rnu0_0, al
printChar rnu0_1
printChar rnu0_0
printChar 58h
printChar 5eh
printChar 31h
printChar 20h
printChar 2bh
printChar 20h
printChar 43h
```

```
            printChar 10d
            printString msg21
            mov ah, 01h
            int 21h
            jmp menu

            fail4_4:
                printChar 10d
                printString msg24
                mov ah, 01h
                int 21h
                jmp menu
```

Esta es la última opción del menú principal, donde se finaliza la ejecución del programa con la instrucción 4c00h con la interrupción 21h, la cual al ejecutarse también le devuelve el cursor al usuario para que continúe con el uso de la consola.

```
opcion8:
            printChar 10d
            printString msg22
            printChar 10d
            mov ax,4c00h;Salir del programa
            int 21h
```

A continuación se muestran los procedimientos utilizados en el programa, donde “cleanScr” sirve para limpiar la pantalla o consola de selecciones previas, como también posicionando el cursor al inicio para que las impresiones en consola no se corran.

```
cleanScr proc;Limpiar consola
        mov ah, 06h
        mov al, 00h
        mov bh, 12h;Color de pantalla, color de texto
        mov cx, 0000h
        mov dx, 184fh
        int 10h

        mov ah, 02h
        mov bh, 00h
        mov dx, 0000h
        int 10h
        ret
    cleanScr endp
```

Con este procedimiento se reinician las variables en caso de que el usuario no ingrese algún valor valido.

```
restartVariables proc;Reiniciar variables en caso de caracter no valido
    mov grade, 00h
    ret
restartVariables endp
```

Este procedimiento verifica si el carácter al momento de ingresar un signo es el carácter correcto y no se ingresa algo que no corresponde.

```
signProof proc;Solicitar signo y verificarlo
    mov ah, 01h
    int 21h
    cmp al, 2bh
    je notFails
    cmp al, 2dh
    notFails:
    ret
signProof endp
```

Por último se tiene el siguiente procedimiento, al igual que el pasado sirve para verificar los caracteres ingresados, solo que este sirve para verificar únicamente números enteros.

```
numberProof proc;Solicitar numero y verificarlo
    mov ah, 01h
    int 21h
    sub al, 30h
    cmp al, 0d
    je notFails2
    cmp al, 1d
    je notFails2
    cmp al, 2d
    je notFails2
    cmp al, 3d
    je notFails2
    cmp al, 4d
    je notFails2
    cmp al, 5d
    je notFails2
    cmp al, 6d
    je notFails2
    cmp al, 7d
    je notFails2
```

```
        cmp al, 8d
        je notFails2
        cmp al, 9d
        notFails2:
        ret
    numberProof endp
```

PROGRAMAS NECESARIOS

- DOSBox 0.74-3
  - Requisitos

    El requisito principal es un puerto de libsdl para su sistema. DOSBox debe compilar en todos los sistemas con un compilador C ++ decente como GCC. Los requisitos mínimos para el puerto Win32 para ejecutar son requisitos mínimos para Windows 9x y tarjeta SVGA. Pero los requisitos del sistema para un juego suave dependen de los requisitos del juego concreto y del hardware elegido para ser emulado en DOSBox.

    Permite ejecutarse en múltiples sistemas operativos definidos y especificados como Linux, FreeBSD, Windows, Mac OS X, OS/2, Palm OS, RISC OS, BeOS, entre otros. También se adaptó a las consolas portátiles PSP y GP2X.

- MASM611
  - Requisitos

    El macro ensamblador de Microsoft es capaz de ensamblar programas para que corran en los microprocesadores 8086-80486 y para sistemas que utilicen el coprocesador matemático. Este ensamblador (MASM versión 6.11) requiere la versión de DOS 3.3 y posteriores implementaclones y un mínimo de 840 Kbytes de memoria de disco, incluidos todos sus archivos adicionales.

SOLUCION DE PROBLEMAS

MARVIN ALEXIS ESTRADA FLORIAN

CORREO: alexis1estrada@hotmail.com

TELÉFONO: +502 3342 1547