Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Laboratorio Arquitectura de Computadoras y Ensambladores 1
Sección “N”
Auxiliar: Robinson Pérez

MANUAL TÉCNICO
PROYECTO 2 FASE 1 Y 2

Nombre y Apellidos
Pedro Antonio Castro Calderón

Carné
201900612

Guatemala 1 de Enero de 2023

DESCRIPCIÓN DE LA APLICACIÓN

La aplicación consiste en un programa creado en lenguaje ensamblador x86, que será capaz de recibir una función de grado n (no mayor a 5) , para posteriormente poder visualizarla en pantalla, resolver su respectiva derivada e integral con la opción de imprimirla también.

HERRAMIENTAS UTILIZADAS

Sistema Operativo: Elementary OS 6.1 Jólnir

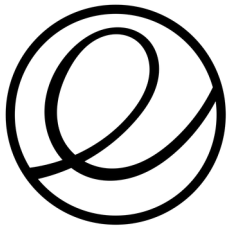
RAM: 8GB

IDE: Visual Studio Code 1.74.1

Ensamblador Utilizado: NASM x86

Consola: Terminal de Linux

Emulador: DOSBox 0.74 -3



ESTRUCTURA DEL PROYECTO

Todo el código está en un solo archivo, llamado Entrada.asm, utilizando la sintaxis de Intel con NASM x86.

En el código, se utilizaron los siguientes registros:

- Registros de 8 bits : al, ah, bl, bh, cl, ch, dl, dh
- Registros de 16 bits: ax, bx, cx, dx

Interrupciones:

Se utilizaron interrupciones de DOS (int 21h) como por ejemplo

- 09h : Imprimir en pantalla
- 01h: Leer un carácter ingresado por el usuario y desplegarlo
- 02h: Despliega un carácter en la pantalla

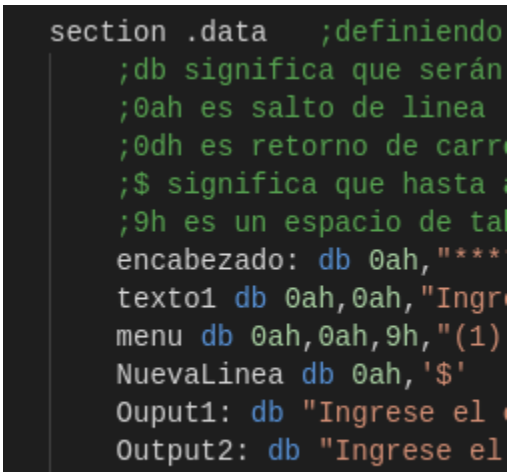
Se utilizaron interrupciones de BIOS (int 10h) como por ejemplo

- 00h: Poner el modo video
- 03h Obtener la posición y el tamaño del cursor

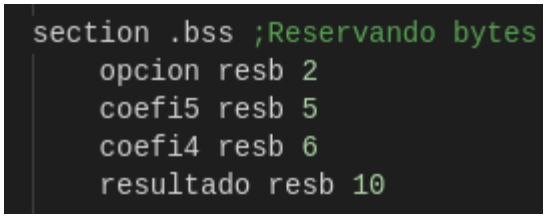
Secciones del código

El código está dividido en 3 secciones:

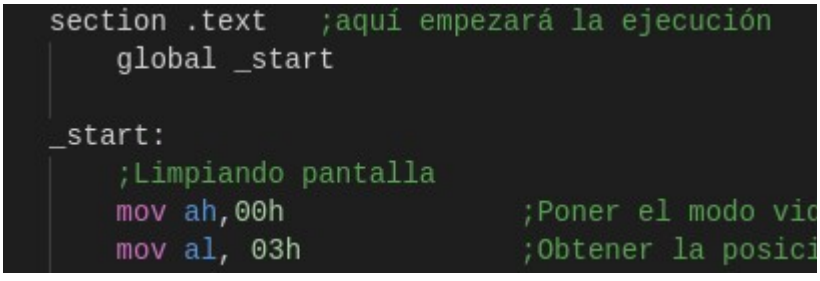
- **section .data:** Esta sección se usa para declarar datos inicializados o constantes. Aquí se pueden declarar valores, nombres de archivo, tamaños de buffer, etc.



- **section .bss:** La sección bss es utilizada para declarar variables, para usarlos en el futuro durante la ejecucioń del programa.



- **section .text:** En esta sección va el código como tal, Esta sección debe empezar con la declaración *global _start* , lo cuál, le dice al kernel dónde comienza la ejecución del programa.



Comentarios

Los comentarios en el lenguaje ensamblador empiezan con un punto y coma (;) , en el código fuente se hizo uso de comentarios para especificar las acciones del código, por ejemplo:

; Esta interrupción muestra un mensaje en la pantalla

Procedimientos:

Almacenan código y son llamados con la palabra reservada *call* , para la creación de la aplicación, se crearon los siguientes procedimientos

- **_getEntrada:** Recibe la opción de menú dónde el usuario desea dirigirse, si existe, lo redirigirá a esa opción.
- **_ingresarCoeficientes:** Procedimiento para ingresar los coeficientes de la ecuación
- **_verFuncion:** Despliega en pantalla la última función almacenada en memoria.
- **_noEsNumero:** Devuelve un mensaje de error al detectar que el usuario ingreso algo distinto a números enteros
- **_imprimirDerivada:** Resuelve la derivada de la función almacenada y la imprime al usuario.
- **_imprimirIntegral**: Resuelve la integral de la función almacenada y la imprime al usuario.
- **_noDisponible:** Muestra un mensaje de que la opción que el usuario ingresó aún no está disponible en la aplicación.
- **_lineaNueva:** Imprime una nueva linea
- **_Salir:** Sale del programa mostrando un mensaje previo a finalizar la ejecución
- **_Escape:** Interrumpe la ejecución del programa.
- **_opcionInvalida:** Muestra un mensaje de que la opción que el usuario ingresó no es válida.

FASE 2

NASM

NASM es el nombre acrónimo de Netwide Assembler. Es un ensamblador de código de bajo nivel para x86 y x86-64 que se utiliza para crear ejecutables para diferentes plataformas como Linux, Windows y MacOS. El lenguaje de ensamblaje es un lenguaje de programación de bajo nivel que se utiliza para escribir código que será ejecutado directamente por el procesador de la computadora. NASM es uno de los ensambladores más comunes y ampliamente utilizados para crear aplicaciones y sistemas operativos de bajo nivel.

NASM fue creado en 1996 por Simon Tatham, quien lo desarrolló como una herramienta de programación de bajo nivel para Linux. Desde entonces, se ha convertido en una de las herramientas de ensamblaje más populares y ampliamente utilizadas para diferentes plataformas y sistemas operativos. Actualmente, NASM está disponible como software libre bajo los términos de la Licencia Pública General de GNU (GNU GPL).

Se decidió utilizar este ensamblador pues se consideró que su sintaxis es bastante sencilla de aprender y entender, además de que el sistema operativo (Linux) utilizado en el proyecto cuenta con un gran soporte para esta herramienta, además de que viene preinstalada por defecto.

DOSBox

DOSBox es un emulador de MS-DOS para sistemas operativos modernos como Windows, Linux y MacOS. Permite a los usuarios ejecutar programas y juegos antiguos que fueron desarrollados para el sistema operativo MS-DOS en sus computadoras actuales.

DOSBox se utiliza a menudo para jugar juegos antiguos de PC que de otra manera no serían compatibles con sistemas operativos modernos. También se puede utilizar para ejecutar aplicaciones antiguas que solo estaban disponibles para MS-DOS. Además, DOSBox se utiliza a menudo en entornos de desarrollo para probar y depurar código de bajo nivel escrito en lenguajes como el ensamblador.

El proyecto debe ejecutarse obligatoriamente en este emulador, debido a que utiliza las interrupciones de DOS en lugar de las del sistema operativo.

Comandos para ejecutar el proyecto desde la terminal de Linux

1. Primero, se compila el archivo con el código en lenguaje ensamblador

nasm -fbin Entrada.asm -o Entrada.com

2. Posteriormente, se llama al programa DOSBox para que ejecute el archivo .com generado anteriormente.

dosbox ./Entrada.com

Esta fase comprende la parte de graficar la función, y resolverla mediante los métodos de aproximación de Newton y Steffensen.

Modo vídeo para graficar

Para activar el modo video se utilizó el modo video 10h , en su variante 13h

Int 10h

Es la forma abreviada de la interrupción 0x10. Esta interrupción controla los servicios de pantalla del PC, se utiliza básicamente para mostrar texto en la pantalla (sin llamar a la INT 21h de MS-DOS o INT 80h de linux), para cambiar a modo gráfico, para establecer la paleta de colores, etc

Modo	Resolución	Colores	Tipo
AL = 00h	40x25	16	Texto
AL = 01h	40x25	16	Texto
AL = 02h	80x25	16	Texto
AL = 03h	80x25	16	Texto
AL = 04h	320x200	4	Gráfico
AL = 05h	320x200	4	Gráfico
AL = 06h	640x200	2	Gráfico
AL = 07h	80x25	2	Texto
AL = 0Dh	320x200	16	Gráfico
AL = 0Eh	640x200	16	Gráfico
AL = 0Fh	640x350	2	Gráfico
AL = 10h	640x350	4	Gráfico EGA 64 KB
AL = 10h	640x350	16	Gráfico EGA menor de 64 KB y VGA
AL = 11h	640x480	2	Gráfico
AL = 12h	640x480	16	Gráfico
AL = 13h	320x200	256	Gráfico

Tabla de modos de video

Modo vídeo 10h:

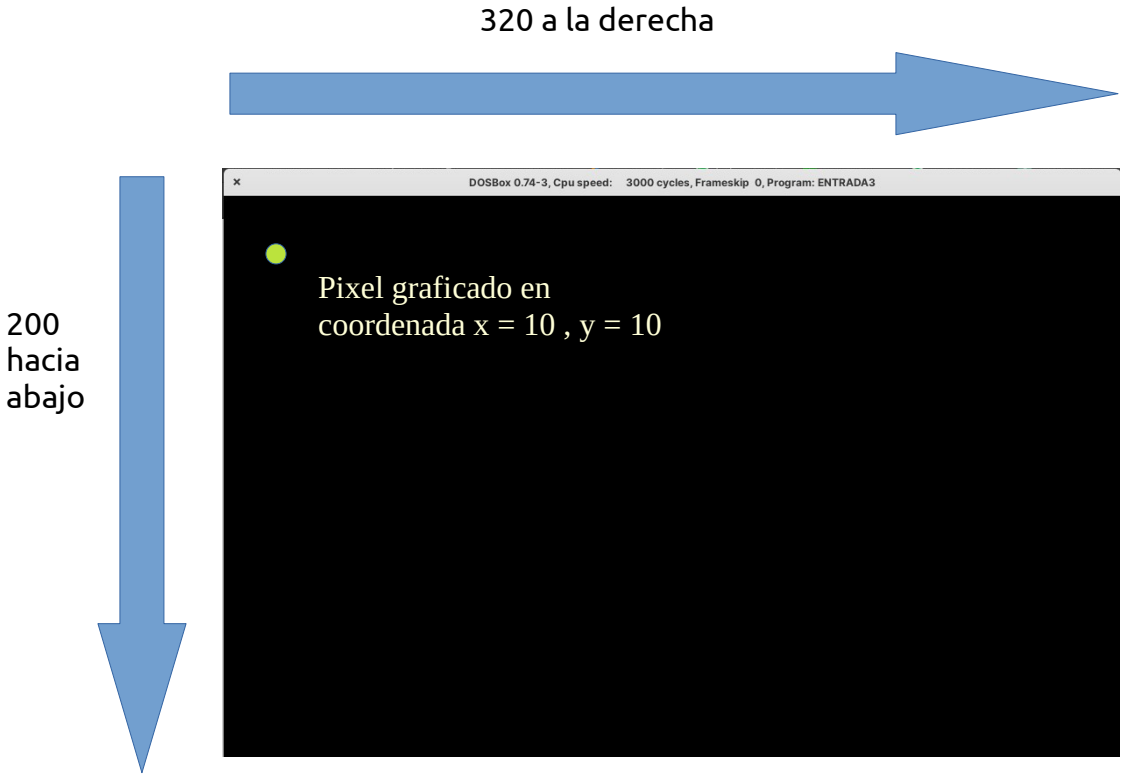
El modo video 10h es un modo de visualización de video utilizado en la arquitectura de computadora x86. En este modo, el sistema operativo puede controlar la forma en que se muestra la información en la pantalla, utilizando una memoria de video especialmente dedicada para almacenar la información a mostrar.

Modo video 13h:

El modo video 13h es similar al modo video 10h, pero permite una mayor resolución de pantalla y un mayor número de colores.

En el modo video 13h, la pantalla se divide en dos áreas: la primera es una zona de texto de 25 líneas por 80 columnas, y la segunda es una zona gráfica de 320 x 200 píxeles. La memoria de video se utiliza para almacenar la información de cada píxel de la pantalla, y el procesador de la computadora accede a la memoria de video para mostrar la información en la pantalla. Este modo fue el que se utilizó en el proyecto para graficar.

Zona gráfica de 320x200



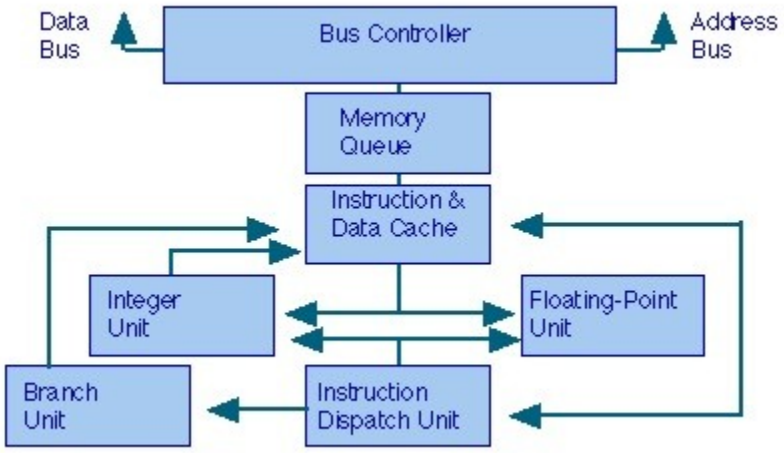
La demostración de arriba muestra solo un píxel, para dibujar lineas continuas, se utilizan ciclos donde en cada iteración, las coordenadas *x* , *y* cambien a la posición deseada, formando la figura que se planea graficar.

Manejando números con decimales

Para el manejo de números que no son enteros, en ensamblador, se hizo uso del FPU.

FPU:

El FPU (Floating Point Unit) es un conjunto de instrucciones y registros en un procesador que se utilizan para realizar operaciones con números de coma flotante. Estos números se utilizan comúnmente para representar valores numéricos con una precisión mayor de la que se puede obtener con números enteros. El FPU está diseñado para ejecutar estas operaciones de forma rápida y eficiente, lo que resulta útil para aplicaciones que requieren una alta precisión en el cálculo de valores numéricos (En este caso, los métodos de Newton y Steffensen).



Instrucciones del FPU utilizadas:

- **fld :** Cargar un número con punto decimal en la pila de registros FPU.

```
fld qword [f_x]
```

- **fild:** Cargar un número entero en la pila de registros FPU.

```
fild word [x_inferior]
```

- **fist:** Copia el valor más reciente de la pila FPU y lo guarda al operando destino (debe ser entero)

```
fistp word [entero]
```

- **fst:** Copia el valor más reciente de la pila FPU y lo guarda al operando destino (debe ser número flotante)

```
fstp qword [f_x]
```

- **fabs:** Le saca el valor absoluto al elemento más reciente de la pila FPU

- **fadd:** Suma el último valor de la pila FPU con el operando a su derecha, y el resultado lo almacena en el último valor de la pila FPU. Si no tiene operando a su derecha, añade el contenido del último valor de la pila al penúltimo valor de la pila FPU.

```
fadd qword [f_x]
```

- **fsub:** Al más reciente valor de la pila FPU se le resta el operando de la derecha y el resultado se guarde en el más reciente valor de la pila.

```
fsub qword [xsub_n]
```

- **fchs:** Cambia el signo del valor más reciente de la pila FPU

- **fdiv:** Divide el más reciente elemento de la pila FPU con el operando de su derecha, y el resultado lo guarda en el elemento más reciente de la pila.

```
fdiv qword [fprima_x]
```

- **fmul:** Multiplica el más reciente elemento de la pila FPU con el operando de su derecha, y el resultado lo guarda en el elemento más reciente de la pila.

```
fmul qword [xsub_n]
```

- **frndint:** Redondea el más reciente elemento de la pila al entero más cercano.

REPOSITORIO DE GITHUB:

https://github.com/PedroCastro2001/ACE1_P2_201900612.git