

Universidad San Carlos de Guatemala
Arquitectura de Computadores y Ensambladores 1
Diciembre 2022

Saul Esteban Castellanos Ubeda
201801178

Proyecto único

Descripción

El proyecto trata de realizar una calculadora gráfica. Los usuarios deben ser capaces de ingresar un polinomio (grado máximo 5), donde cada coeficiente es un número con signo no mayor de 2 dígitos. Para este propósito, se debe utilizar lenguaje ensamblador x86 y un emulador de sistema operativo DOS. El proyecto fue realizado utilizando la sintaxis de MASM.

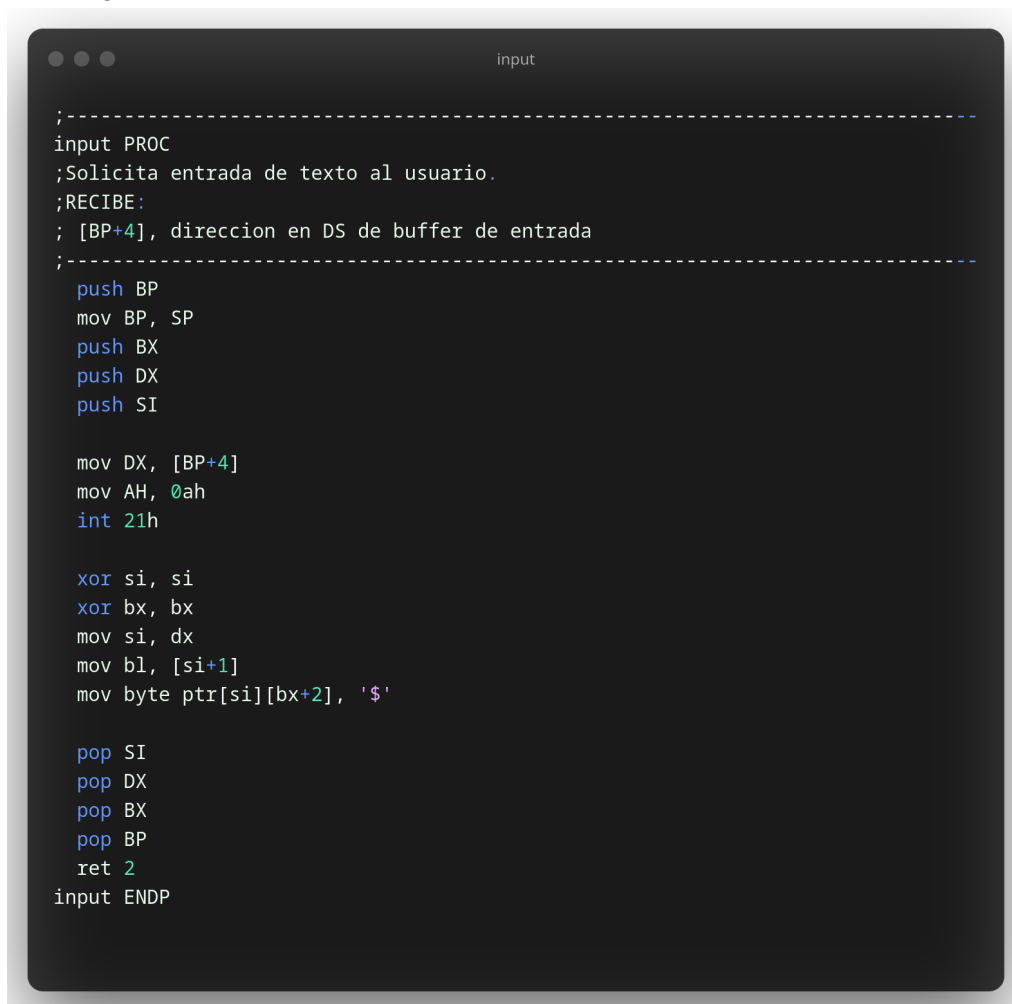
Los objetivos de la primera fase son:

- Solicitar entrada de texto para ingresar los coeficientes de un polinomio
- Mostrar en pantalla el polinomio ingresado
- Obtener la derivada y la integral de dicho polinomio y mostrarlas en pantalla

Entrada de texto

Todas las entradas de texto en el programa son realizadas por medio de la interrupción 21h/0ah.

Para estandarizar la entrada de texto, se desarrolló un procedimiento llamado "input", cuyo código es el siguiente:



```
input
;-----
input PROC
;Solicita entrada de texto al usuario.
;RECIBE:
; [BP+4], direccion en DS de buffer de entrada
;-----
    push BP
    mov BP, SP
    push BX
    push DX
    push SI

    mov DX, [BP+4]
    mov AH, 0ah
    int 21h

    xor si, si
    xor bx, bx
    mov si, dx
    mov bl, [si+1]
    mov byte ptr[si][bx+2], '$'

    pop SI
    pop DX
    pop BX
    pop BP
    ret 2
input ENDP
```

El procedimiento recibe una dirección en memoria de un buffer de entrada como un parámetro de pila. Un ejemplo de buffer de entrada es el que se utiliza para recibir el coeficiente de cada término del polinomio:

```
term_buffer db 4, ?, 4 dup(?)
```

El primer byte indica que el máximo de caracteres es 3, porque el 4to carácter se reserva para recibir el carácter de retorno que indica que se finalizó la entrada.

El segundo byte está sin inicializar porque ahí se almacenará el número de caracteres ingresados en tiempo de ejecución.

Los siguientes 4 bytes son la memoria reservada donde se almacenarán los caracteres en tiempo de ejecución.

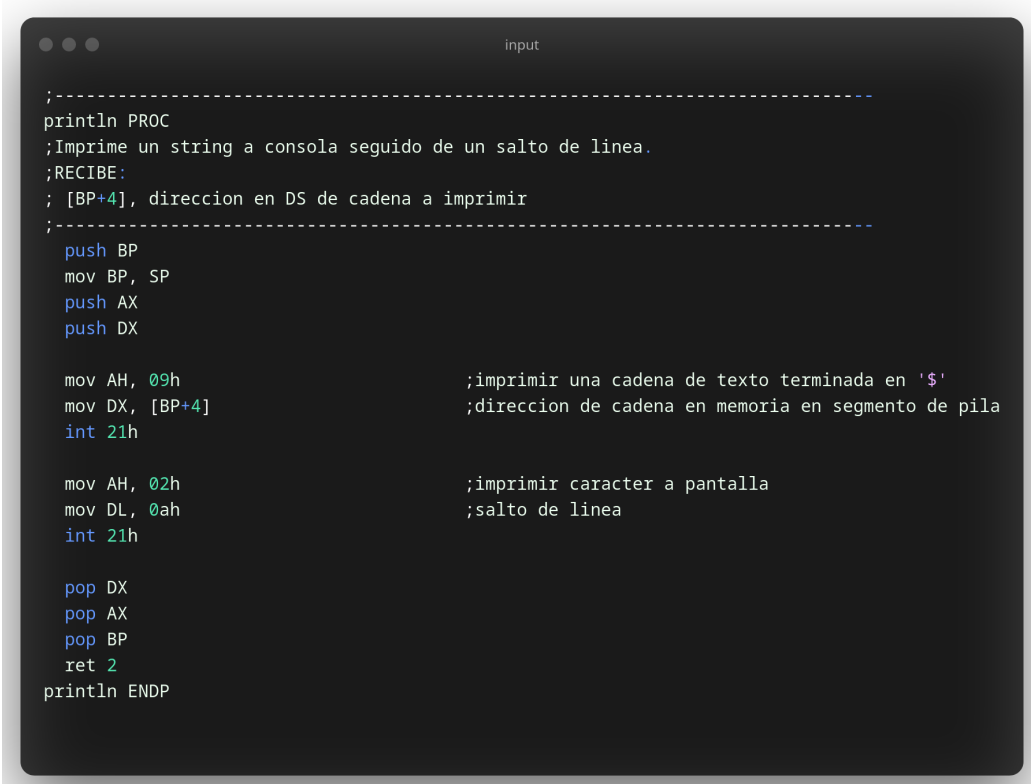
Cada buffer de entrada sigue el mismo formato.

El procedimiento "input" añade un signo '\$' al final del buffer de entrada para que luego pueda ser mostrado en pantalla con la interrupción 21h/09h.

Salida de texto

Para mostrar texto en pantalla se utiliza la interrupción 21h/09h. Esta interrupción permite imprimir en pantalla una cadena de texto almacenada en memoria, siempre y cuando ésta termine con un carácter de '\$'.

Para estandarizar la salida de texto, se realizaron dos procedimientos: "println" y "print", cuya única diferencia es que "println" ingresa un carácter de salto de línea después de la cadena. El código de "println" puede verse en la siguiente imagen:



```
input

;-----
println PROC
;Imprime un string a consola seguido de un salto de linea.
;RECIBE:
; [BP+4], direccion en DS de cadena a imprimir
;-----
    push BP
    mov BP, SP
    push AX
    push DX

    mov AH, 09h                ;imprimir una cadena de texto terminada en '$'
    mov DX, [BP+4]             ;direccion de cadena en memoria en segmento de pila
    int 21h

    mov AH, 02h                ;imprimir caracter a pantalla
    mov DL, 0ah                ;salto de linea
    int 21h

    pop DX
    pop AX
    pop BP
    ret 2
println ENDP
```

Ingreso de términos del polinomio

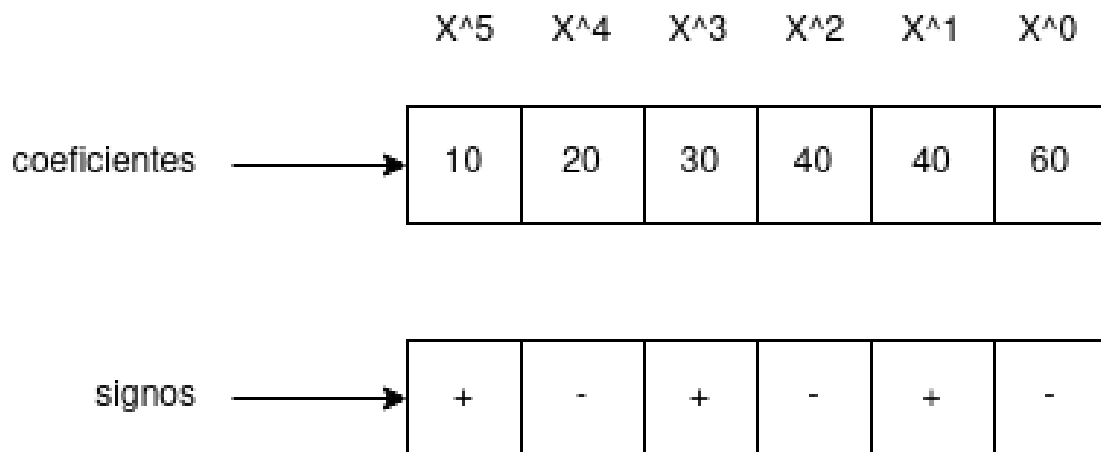
```
-----
inputFunctionTermByTerm PROC
;Entrada de una funcion grado 'n' ingresando cada termino por separado.
;Donde 1 < n < 5
;RECIBE:
; [BP+4], grado de la funcion
;-----
    push bp
    mov bp, sp
    sub sp, 2
    push si
    push di
    push bx
    xor ax, ax
    xor bx, bx
    xor di, di
    xor si, si

    mov ax, 5
    mov bx, [bp+4]           ;Obtener grado de la funcion
    mov word ptr[bp-2], bx   ;Almacenar localmente grado del polinomio
    add word ptr[bp-2], 1    ;Sumarle 1 al grado local
    sub ax, bx               ;5-grado = corrimiento inicio de array de coeficientes
    add si, ax               ;Obtener posicion en array para el coeficiente actual
    xor bx, bx
    mov bx, offset term_msg  ;Mensaje de termino actual
    mov di, lengthof term_msg ;Obtener longitud de mensaje
    sub di, 4                ;Apuntar di hacia 4 caracteres antes del final de cadena
    jmp store_terms

invalid_term:
    push offset regex_msg    ;Avisar que el termino ingresado es invalido
    call println
store_terms:
    mov al, byte ptr[bp+4]    ;Obtener exponente del termino actual
    add al, 30h               ;Convertir exponente en caracter
    mov [bx+di], al           ;Modificar mensaje del termino actual -> "X^{al}"
    push bx                   ;ptr_string = mensaje del termino actual
    call print                 ;print(word ptr_string)
    push offset term_buffer    ;ptr_input_buffer = input buffer para coeficiente
    call input                 ;input(word ptr_input_buffer)
    mov dl, 0ah
    call printChar
    push word ptr term_buffer[1] ;size = longitud del coeficiente(incluyendo signo)
    push offset term_buffer[2]  ;ptr_coefficient = direccion de coeficiente en DS
    call validateCoefficient     ;validateCoefficient(word ptr_coefficient, byte size)
    jnz invalid_term            ;No es coeficiente valido, volver a solicitar
    mov al, term_buffer[2]      ;Obtener signo del coeficiente
    mov byte ptr term_sign[si], al ;Almacenar signo en memoria
    xor ax, ax
    mov al, term_buffer[1]      ;Obtener longitud de coeficiente
    sub al, 1                   ;Restarle 1 a longitud para descartar signo
    push ax                     ;num_digitos = numero de digitos en input buffer
    push offset term_buffer[3]  ;ptr_string = String coeficiente del termino
    call str2num                 ;str2num(word ptr_string, byte num_digitos)
    mov terms[si], al           ;Almacenar coeficiente en array
    inc si                      ;Siguiente posicion de almacenamiento de coeficientes
    dec word ptr[bp+4]
    dec word ptr[bp-2]
    cmp byte ptr[bp-2], 0       ;Ya se obtuvo el ultimo coeficiente?
    jne store_terms             ;No, repetir hasta obtener el ultimo coeficiente

    pop bx
    pop di
    pop si
    mov sp, bp
    pop bp
    ret 2
inputFunctionTermByTerm ENDP
```

Los coeficientes son ingresados uno por uno. Existen dos arreglos en memoria, uno para los signos de cada coeficiente, y otro para almacenar los coeficientes. Debido a que el grado máximo para un polinomio es 5, cada arreglo está conformado por 6 espacios, que corresponden a cada término del polinomio.



Dependiendo del grado del polinomio, se empieza a almacenar los coeficientes a partir de un corrimiento desde el principio del arreglo. Este corrimiento se calcula por medio de la siguiente fórmula:

$$\text{corrimiento} = 5 - \text{grado del polinomio}$$

El almacenamiento de coeficientes en el arreglo está dado por la siguiente fórmula:

$$\text{posición de coeficiente en arreglo} = \text{índice actual} + \text{corrimiento}$$

Para un polinomio grado 3, el corrimiento sería 2. Empezando desde el primer índice, los coeficientes se almacenarían desde el tercer índice.

El almacenamiento de los signos de cada coeficiente sigue la misma lógica.

Dibujar píxeles en pantalla

En este proyecto se utiliza el modo de video 13h, es decir, 320 píxeles de ancho, 200 píxeles de alto y 256 colores. La razón principal de escoger este modo es que fácilmente se puede indexar la memoria de video.

Para mostrar un píxel, simplemente se almacena un número entero en la posición de memoria de video correspondiente a la coordenada de la pantalla. El número entero almacenado hace referencia al índice de la paleta de colores del chip de vídeo.

Para acceder a la dirección de memoria adecuada, se hace un mapeo lexicográfico de la memoria de video.

La fórmula para obtener la dirección es:

$$\text{dirección} = \text{fila} * 320 + \text{columna}$$

Para poder hacer de este proceso más fácil, se utilizó el procedimiento “screenCoord”, cuyo código puede verse a continuación:

```
screenCoord
;-----
screenCoord PROC
;Realiza el calculo de la posicion en pantalla para las coordenadas (columna, fila).
;RECIBE:
; [bp+4], columna
; [bp+6], fila
;RETORNA:
; AX, posicion en memoria de video
;-----
push bp
mov bp, sp

mov ax, SCREEN_WIDTH
mul word ptr [bp+6]
add ax, [bp+4]                ;Posicion = fila * SCREEN_WIDTH + columna

pop bp
ret 4
screenCoord ENDP
```

Graficar un polinomio

El proceso de graficación empieza por encontrar el incremento que debe tener cada valor de X. Para poder determinarlo, se utiliza la siguiente fórmula:

$$\text{incremento_x} = \text{lim_sup} - \text{lim_inferior} / \text{ancho_pantalla}$$

Donde:

- incremento_x: Incremento de X para cada pixel del ancho de la pantalla.
- lim_sup: límite superior del rango de X.
- lim_inf: límite inferior del rango de X.
- ancho_pantalla: El ancho de la pantalla (320 para modo de vídeo 13h)

Esta fórmula permite mapear el rango a lo largo de toda la pantalla.

Los valores que tome f(x) son multiplicados por un factor de escala para que la gráfica pueda visualizarse mejor. Para este proyecto el factor de escala de f(x) se tomó como 10.

Al momento de graficar, se toma el arreglo de coeficientes de un polinomio, y se llama el procedimiento “evalOnX”. Este procedimiento permite evaluar el polinomio en un valor de X, y almacena el valor de f(x) en memoria. El código puede verse a continuación:

```

evalOnX
;-----
evalOnX PROC
;Evalua un polinomio respecto a X.
;RECIBE:
; [bp+4], Exponente inicial
; ds:si, direccion del primer coeficiente
;ENTREGA:
; ds:yVal, resultado de evaluacion
;-----

push bp
mov bp, sp
sub sp, 2
push cx

exponente equ word ptr [bp+4]
coeficiente equ word ptr [bp-2]

fldz                      ;ST(0)=yVal=0.0
load_val:
xor ax, ax
mov al, [si]
cbw                      ;Extender signo de AL hacia AH
mov coeficiente, ax      ;Almacenar coeficiente en variable local
cmp exponente, 0         ;El exponente es 0?
jz sum_coeff             ;Si, solamente sumar coeficiente al total
fld qword ptr xVal       ;No, ST(0)=xVal. Cargar xVal al fpu
; ST(1)=yVal
mov cx, exponente        ;Cargar exponente inicial
jmp xVal_to_power        ;Saltar hacia la parte de exponenciacion
sum_coeff:
fld coeficiente          ;Cargar coeficiente al fpu
jmp next_term            ;Saltar hacia la parte de la suma

xVal_to_power:
dec cx                   ;Disminuir exponente
cmp cx, 0                ;Exponente 0?
jz end_power             ;Si, Seguir con evaluacion del termino
fmul qword ptr xVal      ;No, ST(0)=ST(0)*xVal
; ST(1)=yVal
jmp xVal_to_power

end_power:
fimul coeficiente        ;ST(0)=ST(0)*coeficiente
;ST(1)=yVal

next_term:
fadd                     ;ST(0)=vaciado
;ST(1)=ST(0)+ST(1). ST(1) es nuevo ST(0)
inc si                   ;Apuntar a siguiente coeficiente
dec exponente            ;Exponente del siguiente termino
cmp byte ptr[si], 'd'    ;Ya llegamos al final del arreglo de coeficientes?
jnz load_val             ;No, repetir hasta llegar al final
;Si, almacenar resultado en memoria
fstp qword ptr yVal      ;ST(0)=vaciado

pop cx
mov sp, bp
pop bp
ret 2
evalOnX ENDP

```

Para realizar el cálculo de $f(x)$, es necesario hacer uso de números decimales. Se utiliza el FPU para operar los números decimales. Para el redondeo se escogió el método de truncar.