Práctica 1: Serie de coseno para x número[*]

Cindy Melissa, Gatica Arriola 201709692[1,**]

[1] *Facultad de Ingeniería, Universidad de San Carlos, Ciudad Universitaria, Zona 12, Guatemala.*

En el proyecto de codificación en lenguaje ensamblador ARM para calcular la serie de coseno, se implementó la serie de Taylor utilizando sumatorias para aproximar el valor del coseno de un ángulo dado en radianes. Se escribió un código que define el ángulo de entrada, inicializa las variables necesarias y utiliza un bucle para calcular cada término de la serie y sumarlos, aplicando las fórmulas matemáticas correspondientes. El código incluye operaciones aritméticas y de control de flujo específicas del lenguaje ensamblador ARM para gestionar los cálculos y el almacenamiento de resultados. Al finalizar, se obtuvo una aproximación precisa del valor del coseno para el ángulo entrada, demostrando la efectividad de la serie de Taylor en cálculos trigonométricos.

I. OBJETIVOS

A. Generales

- Calcular el valor de la función seno para un ángulo dado en radianes utilizando la serie de Taylor.

B. Específicos

* Implementar la serie de Taylor en ensamblador.

* Comprender el uso de registros y operaciones en punto flotante.

* Validar el cálculo del seno comparándolo con valores conocidos.

II. MARCO TEÓRICO

A. Lenguaje Ensamblador

El lenguaje ensamblador (Assembly) es un lenguaje de bajo nivel que se encuentra entre el código máquina y los lenguajes de alto nivel como C o Python. Se caracteriza por ser específico para una arquitectura de procesador en particular y permite un control detallado del hardware. Algunas características clave del lenguaje ensamblador incluyen:

1. **Bajo nivel de abstracción:** cada instrucción en ensamblador representa directamente una operación del hardware.

2. **Alto rendimiento:** al escribir código optimizado, se pueden lograr tiempos de ejecución más rápidos en comparación con lenguajes de alto nivel.

3. **Uso directo de registros:** se trabaja manipulando directamente los registros del procesador, en lugar de variables abstractas.

4. **Complejidad y dificultad de mantenimiento:** debido a su bajo nivel, el código en ensamblador es difícil de leer, escribir y mantener.

El proyecto consiste en implementar el juego "Simón dice", donde el usuario debe recordar un patrón de colores que se va extendiendo en cada ronda. La lógica de implementación en ensamblador implica varios aspectos:

1. **Interacción con los pines GPIO:** se deben controlar LEDs de distintos colores conectados a los pines GPIO de la Tiva C y será necesario configurar los registros adecuados para habilitar los pines como salida.

2. **Generación de Secuencia de Colores:** se podría almacenar y manejar una secuencia creciente de colores que el usuario debe repetir. Además, se pueden utilizar registros o memoria RAM para almacenar los valores.

3. **Interrupciones y Temporización:** se pueden emplear retardos para mostrar cada color durante un tiempo determinado o utilizar temporizadores (TIMER) de la Tiva C para manejar la duración de los flashes de luz.

4. **Captura de Entrada del Usuario:** con botones para que el usuario ingrese la secuencia capturando la entrada de los botones mediante polling o interrupciones.

B. Tiva C Series

La Tiva C Series (TM4C123G) es un microcontrolador basado en la arquitectura ARM Cortex-M4, que admite instrucciones Thumb-2, combinando eficiencia y rendimiento. En ensamblador, se trabaja directamente con registros y periféricos para optimizar el control del hardware.

[*] Laboratorio Electronica 5

[**] 2787947930101@ingenieria.usac.edu.gt

Para implementar el juego "Simón dice", se podría configurar los GPIO para controlar los LEDs y detectar la pulsación de botones, almacenando la secuencia de colores en registros o RAM. O con temporizadores para los retardos en la visualización de los LEDs y polling o interrupciones para capturar la entrada del usuario. Finalmente, el código en ensamblador comparará la secuencia ingresada con la generada, determinando si el usuario avanza o falla.

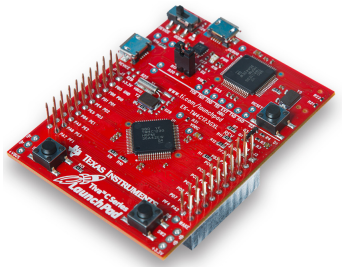

Figura 1: Tiva C Series

C. Serie de Taylor del seno

La serie de Taylor para la función seno es una representación matemática que permite aproximar el valor del seno de un ángulo a través de una suma infinita de términos basados en las derivadas de la función en un punto específico. En el caso del seno, la serie se expande alrededor del punto $x = 0$ La fórmula general para la serie de Taylor del seno es:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad , \forall x$$
$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad , \forall x$$

Figura 2: Serie de Taylor del seno

III. DISEÑO EXPERIMENTAL

A. Materiales y Herramientas

- **Software:** Ensamblador ARM- Keil uVision.
- **Hardware:** Microcontrolador TM4C123GH6PM.

B. Descripción del código

1. *Registros y variables*

- **S0:** Almacena el ángulo en radianes.
- **S6:** Número de iteraciones.
- **S2, S3, S4, S5, S7, S9, S10, S11, S13, S16:** Registros auxiliares para cálculos intermedios.

2. *Funciones*

- sumatoria: Controla el ciclo de iteraciones.
- factorial: Calcula el factorial de un número.
- potenciaa: Calcula la potencia $(-1)^n$.
- potenciab: Calcula la potencia $x^{(2n+1)}$.

C. Procedimiento Experimental

- **Paso 1:** Cargar el ángulo y el número de iteraciones.
- **Paso 2:** Iniciar el ciclo de iteraciones.
- **Paso 3:** En cada iteración, calcular el término actual de la serie de Taylor.
- **Paso 4:** Acumular el resultado parcial.
- **Paso 5:** Comparar el resultado final con valores de referencia.

IV. DISCUSIÓN

Se implementa la serie de Taylor para calcular la función seno de un ángulo medido en radianes. El código inicializa varias constantes y variables para manejar las iteraciones y cálculos. Se utiliza un bucle principal para realizar la sumatoria, incrementando el número de iteración, calculando el factorial del denominador y las potencias del numerador, y sumando los términos resultantes. Además, se definen funciones auxiliares para calcular el factorial, la potencia del numerador $(-1)^n$ y la potencia del ángulo $x^(2n + 1)$. El bucle se repite hasta completar el número especificado de iteraciones, almacenando el resultado final en el registro S9. Al finalizar, el resultado del cálculo se ajusta sumándole el valor del ángulo x, proporcionando así el valor del seno del ángulo dado.

V. RESULTADOS

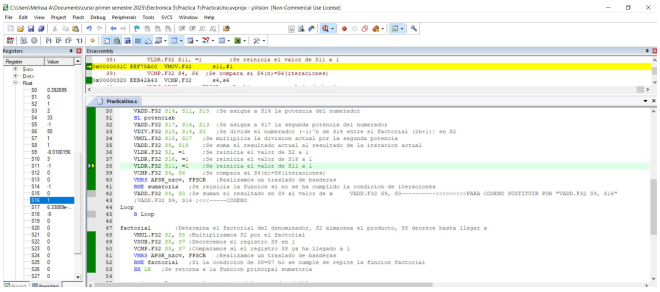

Figura 4: Código en Keil Vision

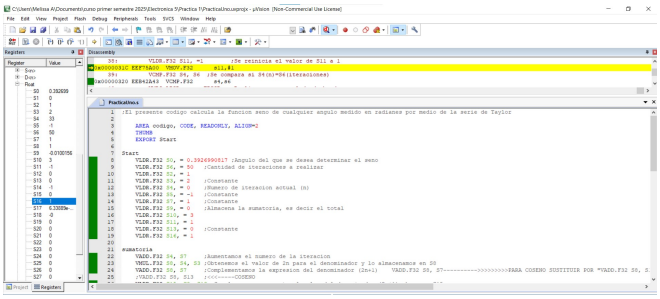

Figura 3: Código en Keil Vision

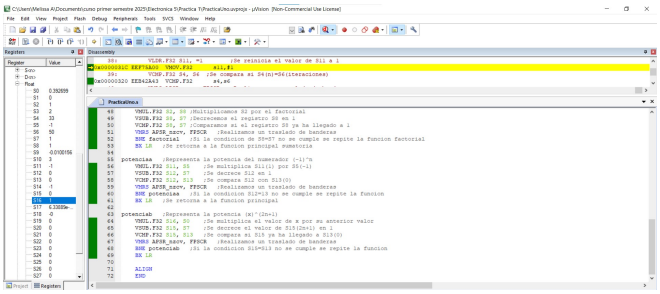

Figura 5: Código en Keil Vision

[1] Santos, Hugh D. y Melissinos (Décimo tercera edición). (2013). *Lenguaje ensamblador para principiantes*. México, Pearson.

[2] Steven Barco Tangarife.*Simón dice Arduino* Ingenio, 2017.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE MECANICA ELÉCTRICA
MICROCONTROLADORES

INTEGRANTES:

No.	NOMBRE	CARNÉ
1	Cindy Melissa Gatica Arriola	201909692
2		
3		
4		
5		

PRACTICA No. 1
Serie de Seno y Coseno

```
;El presente codigo calcula la funcion seno de cualquier angulo medido en radianes por medio de la serie de Taylor

        AREA codigo, CODE, READONLY, ALIGN=2
        THUMB
        EXPORT Start

Start
        VLDR.F32 S0, = 0.3926990817    ;Angulo del que se desea determinar el seno
        VLDR.F32 S6, = 50       ;Cantidad de iteraciones a realizar
        VLDR.F32 S2, = 1
        VLDR.F32 S3, = 2        ;Constante
        VLDR.F32 S4, = 0        ;Numero de iteracion actual (n)
        VLDR.F32 S5, = -1       ;Constante
        VLDR.F32 S7, = 1        ;Constante
        VLDR.F32 S9, = 0        ;Almacena la sumatoria, es decir el total
        VLDR.F32 S10, = 3
        VLDR.F32 S11, = 1
        VLDR.F32 S13, = 0       ;Constante
        VLDR.F32 S16, = 1

sumatoria
        VADD.F32 S4, S7                 ;Aumentamos el numero de la iteracion
```

```
        VMUL.F32 S8, S4, S3 ;Obtenemos el valor de 2n para el denominador y lo almacenamos en S8
        VADD.F32 S8, S7             ;Complementamos la expresion del denominador (2n+1)    VADD.F32 S8,
S7---------->>>>>>>>>>PARA COSENO SUSTITUIR POR "VADD.F32 S8, S13"
        ;VADD.F32 S8, S13       ;<<<-----COSENO
        VADD.F32 S15, S8, S13 ;Se almacena nuevamente el valor del denominador (2n+1) ahora en S15
        BL factorial            ;Salta a la funcion de factorial
        VADD.F32 S12, S4, S13   ;Se asigna el valor de n a S12 para la funcion potenciaa
        BL potenciaa
        VADD.F32 S14, S11, S13 ;Se asigna a S14 la potencia del numerador
        BL potenciab
        VADD.F32 S17, S16, S13 ;Se asigna a S17 la segunda potencia del numerador
        VDIV.F32 S18, S14, S2   ;Se divide el numerador (-1)^n de S14 entre el factorial (2n+1)! en S2
        VMUL.F32 S18, S17       ;Se multiplica la division actual por la segunda potencia
        VADD.F32 S9, S18        ;Se suma el resultado actual al resultado de la iteracion actual
        VLDR.F32 S2, =1         ;Se reinicia el valor de S2 a 1
        VLDR.F32 S16, =1        ;Se reinicia el valor de S16 a 1
        VLDR.F32 S11, =1        ;Se reinicia el valor de S11 a 1
        VCMP.F32 S4, S6             ;Se compara si S4(n)=S6(iteraciones)
        VMRS APSR_nzcv, FPSCR       ;Realizamos un traslado de banderas
        BNE sumatoria  ;Se reinicia la funcion si no se ha cumplido la condicion de iteraciones
        VADD.F32 S9, S0         ;Se suman el resultado en S9 al valor de x     VADD.F32 S9, S0---------->>>>>>>>>PARA
COSENO SUSTITUIR POR "VADD.F32 S9, S16"
        ;VADD.F32 S9, S16 ;<<<-----COSENO
Loop
        B Loop

factorial               ;Determina el factorial del denominador, S2 almacena el producto, S8 decrece hasta llegar a
        VMUL.F32 S2, S8         ;Multiplicamos S2 por el factorial
        VSUB.F32 S8, S7         ;Decrecemos el registro S8 en 1
        VCMP.F32 S8, S7         ;Comparamos si el registro S8 ya ha llegado a 1
        VMRS APSR_nzcv, FPSCR       ;Realizamos un traslado de banderas
        BNE factorial     ;Si la condicion de S8=S7 no se cumple se repite la funcion factorial
        BX LR    ;Se retorna a la funcion principal sumatoria

potenciaa       ;Representa la potencia del numerador (-1)^n
        VMUL.F32 S11, S5        ;Se multiplica S11(1) por S5(-1)
        VSUB.F32 S12, S7        ;Se decrece S12 en 1
        VCMP.F32 S12, S13       ;Se compara S12 con S13(0)
        VMRS APSR_nzcv, FPSCR       ;Realizamos un traslado de banderas
        BNE potenciaa   ;Si la condicion S12=13 no se cumple se repite la funcion
        BX LR    ;Se retorna a la funcion principal

potenciab       ;Representa la potencia (x)^(2n+1)
        VMUL.F32 S16, S0        ;Se multiplica el valor de x por su anterior valor
        VSUB.F32 S15, S7        ;Se decrece el valor de S15(2n+1) en 1
```

```
VCMP.F32 S15, S13         ;Se compara si S15 ya ha llegado a S13(0)
VMRS APSR_nzcv, FPSCR             ;Realizamos un traslado de banderas
BNE potenciab   ;Si la condicion S15=S13 no se cumple se repite la funcion
BX LR

ALIGN
END
```