UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

INFORME DE LABORATORIO 2: ACERCÁNDOSE AL HARDWARE: PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Nombre: John Serrano C.
Ayudante: Maximiliano Orellana A.
Asignatura: Arquitectura de Computadores

07 de diciembre 2021

Tabla de Contenidos

1. **Introducción**……2
   - **1.1 Descripción del Problema**……2
   - **1.2 Herramientas utilizadas en la solución de cada problema**……3
   - **1.3 Objetivos de la experiencia**……3
2. **Marco teórico**……3
3. **Desarrollo de la solución**……5
   - **3.1 Solución problema 1**……5
   - **3.2 Solución problema 2**……6
   - **3.3 Solución problema 3 parte A**……6
   - **3.4 Solución problema 3 parte B**……7
   - **3.5 Solución problema 4 parte A – Función Seno**……7
   - **3.6 Solución problema 4 parte A – Función Logaritmo Natural**……8
   - **3.7 Solución problema 4 parte B**……9
4. **Resultados**……9
5. **Conclusiones**……11

1. INTRODUCCIÓN

El presente informe corresponde al laboratorio número 2 del curso de Arquitectura de Computadores. Este segundo laboratorio se compone en distintos problemas matemáticos, donde se busca obtener soluciones utilizando el lenguaje ensamblador MIPS a través del IDE MARS, usando distintas herramientas de este como son las subrutinas, syscalls, entre otras cosas. El informe sigue la siguiente estructura: Primero, se tiene una descripción breve del problema, los objetivos de la experiencia y las herramientas utilizadas en cada problema. Luego de eso se tiene un marco teórico, donde se explican los conceptos más importantes necesarios para entender la experiencia. Posterior a eso se explica brevemente la solución de cada problema junto con el proceso de ideación de esta, para luego pasar a los resultados obtenidos de cada problema, junto con algunos ejemplos de muestra. Finalmente se tiene una conclusión a la experiencia realizada donde se resume el trabajo realizado y se comprueban los objetivos, junto con otras cosas.

1.1 DESCRIPCIÓN DE CADA PROBLEMA

La experiencia consiste en 4 partes. A continuación, se define cada una, describiendo también sus distintas partes para aquellos partes que tienen una parte A y una parte B.

Problema 1: Se requiere de un programa que calcule el máximo entre dos números enteros. Para ello, se requiere que se reciban los números como entrada y se muestren los mensajes correspondientes para cada número de entrada y para la salida, esto a través del uso de “syscall”. También se requiere que el cálculo del número máximo sea realizado en una subrutina.

Problema 2: Se requiere de un programa que calcule el Máximo Común Divisor (MCD) entre dos números enteros. El programa debe usar una subrutina con recursión, realizando una forma recursiva del Algoritmo de Euclides. Los números deben estar “en duro” en el código.

Problema 3: Tiene dos partes. La parte A, requiere de un programa que realice la multiplicación de dos números enteros, utilizando subrutinas y operaciones matemáticas. La parte B es bastante similar a la parte A, solo que en este caso se busca un programa que realice la división de dos números enteros. Para esta parte, los decimales del resultado como mínimo deben ser 2. Para ambas partes se prohíbe el uso de instrucciones como mul, mult, div, rem, sll, sra, srl y variantes de estas.

Problema 4: Tiene dos partes. Para la parte A, se requiere de dos programas, uno donde se calcule una aproximación a la función Sin(x) y otro donde se calcule una aproximación a la función Ln(x). Para ello se deben utilizar expansiones de Taylor en torno a 0 (Serie de Maclaurin) y deben ser utilizadas las multiplicaciones y divisiones creadas en el problema 3. Para la parte B, se requiere de un grafico que muestre el error de la parte A, evaluando con 3 números distintos y utilizando la expansión de orden 1 hasta orden 11.

1.2 HERRAMIENTAS UTILIZADAS EN LA SOLUCIÓN DE CADA PROBLEMA

A continuación, se mencionan las "herramientas" utilizadas para construir la solución de cada problema:

- **Problema 1:** Uso de syscalls, subrutinas, etiquetas.
- **Problema 2:** Uso de syscalls, subrutinas, stack, recursión y etiquetas.
- **Problema 3 A y B:** Uso de syscalls, subrutinas, registros del coprocesador 1 y etiquetas.
- **Problema 4 Parte A:** Uso de syscalls, subrutinas, registros del coprocesador 1 y etiquetas.

Todos los conceptos mencionados anteriormente son descritos en el marco teórico.

1.3 OBJETIVOS DE LA EXPERIENCIA

Como **objetivos principales** de la experiencia se tienen:

- Aprender sobre el Lenguaje Ensamblador MIPS y su uso a través del IDE MARS.
- Conocer y utilizar distintas herramientas del lenguaje, como son el uso de syscalls, subrutinas, instrucciones de salto, entre otras.
- Implementar algoritmos para resolver los problemas mencionados al inicio de la introducción.

Como **objetivos secundarios:**

- Mejorar la capacidad de entender el funcionamiento de un lenguaje ensamblador.
- Programar una solución correcta utilizando buenas prácticas de la programación.

2. MARCO TEORICO

A continuación, se describen brevemente los conceptos más importantes necesarios para entender completamente la experiencia realizada:

Lenguajes de programación: Corresponden a un conjunto de instrucciones mediante las cuales los usuarios interactúan con el computador, el cual gracias a estas instrucciones y/o información, puede realizar distintas tareas. Existen distintos tipos de lenguaje de programación.

Lenguaje ensamblador: Es un lenguaje de programación, con la característica de que esta diseñado para interactuar con un procesador en específico. Es denominado como "lenguaje maquina".

MIPS: Corresponde a un lenguaje ensamblador. Sus siglas son "*Microprocesor without Interlocked Pipelined Stages*", lo cual se traduce literalmente a "Micoprocesador sin etapas

canalizadas interconectadas". MIPS tiene distintas características, como son los registros, subrutinas, stack, instrucciones de salto, entre otras características.

MARS: Corresponde a un IDE para MIPS. Un IDE, es un entorno de desarrollo integrado, el cual permite al usuario poder interactuar y crear código con un lenguaje de programación deseado.

Registros: En simples palabras, son los "lugares" donde se van guardando distinta información. Algunos registros solo guardan una especie de información en especifica y otros no son utilizables por el usuario. Por buenas prácticas, los registros temporales (registros $t) deben ser limpiados antes de finalizar el programa.

Syscalls: Tal como lo dice su nombre, son llamadas del sistema. Permiten recibir entradas por consola e imprimir salidas en esta. Dependiendo de lo que se quiere recibir o imprimir, se tienen distintos códigos, pero los más importantes para esta experiencia son:

- li $v0, 1: Se prepara para imprimir un número entero.
- li $v0, 2: Se prepara para imprimir un número flotante.
- li $v0, 4: Se prepara para imprimir un string o cadena de texto.
- li $v0, 5: Se prepara para recibir un número entero por consola.
- li $v0, 6: Se prepara para recibir un número flotante por consola.

Subrutinas: Corresponde a una parte pequeña de un algoritmo que se difiere del algoritmo principal debido a que se encarga de realizar una tarea en específico. Por buenas prácticas, los parámetros de entrada de las subrutinas se guardan en los registros $a y las salidas de estas se guardan en los registros $v.

Stack: Corresponde a una memoria, donde se va guardando información importante, como son algunas direcciones o datos que el programador desee respaldar. El uso de stack va de la mano con el uso de recursión. Por buenas prácticas, el stack debe ser limpiado antes de finalizar el programa.

Recursión: En un proceso que permite que una subrutina se llame a si misma, realizando procesos con repetición para luego devolverse y obtener un resultado en especifico una vez que se acaben las repeticiones. El uso de recursión va de la mano con el stack, pues es ahí donde se van guardando los valores temporales y direcciones de memoria durante el proceso recursivo.

Números Enteros: Corresponden a aquellos números positivos y negativos que no tienen una parte decimal. El máximo entre dos números se determina gracias a su orden en la recta de los números enteros, la cual va desde el infinito negativo hasta el infinito positivo, con el número 0 al medio de la recta.

Números Flotantes: Corresponden a aquellos números que tienen una parte decimal. En MIPS, estos números se guardan en los registros $f del coprocesador 1

Coprocesador 1: Corresponden a todos los registros $f, donde se guardan números flotantes. En los registros pares (incluyendo a $f0) es posible guardar números de tipo double, aunque para esta experiencia no son necesarios aquellos números.

Máximo Común Divisor (MCD): Corresponde al número entero mayor que divide a dos números sin dejar un resto. Independiente de si los números enteros son negativos, el MCD siempre es positivo. Una forma bastante eficiente de encontrar el MCD es a través del Algoritmo de Euclides.

Algoritmo de Euclides: Es un método para encontrar rápidamente el MCD entre dos números. Consiste en hacer la división entre ambos números (dejando como dividendo al mayor), luego usar al dividendo como divisor y el resto de la división anterior como divisor. Repetir el proceso hasta que el resto sea 0 y el ultimo divisor corresponderá al MCD.

Función Seno: Corresponde a una función trigonométrica, la cual puede ser aproximada gracias a su serie de Taylor. Esta definida para todos los números enteros.

Función Logaritmo Natural: Corresponde a una función que puede ser aproximada gracias a su serie de Taylor, aunque la serie de Taylor solo tiene una aproximación eficiente para $|x| < 1$. La función solo permite valores mayores que 0.

Serie de Taylor: Corresponde a una serie de potencias que se prolonga hasta el infinito. Se puede cambiar el orden para que esta no vaya al infinito, dando distintos resultados, ya que mientras mayor sea el orden, mayor será la cercanía al valor real de la función a evaluar.

3. DESARROLLO DE LA SOLUCIÓN

Cabe mencionar que en cada problema se respetaron las buenas prácticas, limpiando los registros temporales al final de cada programa y utilizando los registros correspondientes para las subrutinas, a excepción del problema 4A, donde se optó por utilizar otros registros como lo son los registros $f19, $f22, $f24 y $f25 para reemplazar a los registros $a, Y los registros $f4 y $f5 para reemplazar a los registros $v, debido a que se adaptaron todas las subrutinas para que reciban y retornen números flotantes

Para cada problema, se diseñó la siguiente solución, la cual se explicará **brevemente**:

3.1 SOLUCIÓN PROBLEMA 1

Para saber qué número es mayor que otro, se debe hacer una comparación. Por lo tanto, primero, se parte pidiendo los números a comparar por consola, junto con sus respectivos mensajes. En la subrutina “comparación” se comprueba si $a1 < $a2. Si no se cumple lo anterior, se realiza un salto a la etiqueta “primeroEsMayor”, donde se designa a $a1 como retorno de la subrutina. En caso contrario, no se realiza el salto y se designa a $a2 como

retorno de la subrutina. Se finaliza la subrutina y finalmente, el programa imprime el mensaje final junto con el resultado obtenido y se finaliza el programa.

Fue bastante fácil idear la solución pues solo requiere de una comparación.

3.2 SOLUCIÓN PROBLEMA 2

Primero, se cargan los números que se encuentran en la sección .data del programa, y se muestran por consola. Se llama a la subrutina “MCD”, donde lo primero que se hace es hacer espacio para tres números en el stack pointer, ya que es necesario para el proceso recursivo. Se verifica si el segundo número es 0 lo que implica que ya se llegó al final del proceso del Algoritmo de Euclides” y se realiza un salto para terminar con el proceso. Si no se realiza el salto, se copia el segundo número y se realiza la división entre el primer número y el segundo. Luego se utiliza la instrucción “mfhi” para obtener el resto de la división y copiarlo al registro $a2, pues el resto se guarda en el registro hi. Se vuelve a llamar a la propia subrutina y el proceso se repite hasta que $s1 sea 0. Luego del salto, se copia el contenido de $s0 a $v1, para que no se pierda el resultado del algoritmo y se recuperan y limpian los valores del stack, junto con el espacio reservado. Finalmente se realiza un jump register y la recursión se desenvuelve hasta que se vuelve a main. Aquí se comprueba si el MCD dio negativo y si es así, se salta a “cambiarSigno” para cambiar el signo del resultado. Finalmente, se limpian los registros temporales utilizados y se finaliza el programa.

Gracias a la recursión y el stack, se puede traducir el algoritmo de Euclides a un algoritmo en MIPS. El único inconveniente es que a veces el valor termina siendo negativo, pero se corrige al final para evitar inconsistencias.

3.3 SOLUCIÓN PROBLEMA 3 PARTE A

Primero, se cargan los números que se encuentran en la sección .data del programa, y se muestran por consola. Se comprueba si alguno o ambos de los números cargados son negativos, y si es así, se les cambia de signo, pero se deja registro a través de un 1 en los registros $t3 o $t4 si el primer o segundo número era negativo. Luego de eso, se procede a realizar la multiplicación, a través de un ciclo de sumas de uno de los números. El proceso se repite hasta que un contador iguala al segundo número (o el primero, si el segundo se designa para sumar) y se obtiene el proceso de multiplicación. Luego de eso, se verifica si los signos eran iguales o distintos, a través de los 1 guardados en los registros temporales. Si son distintos, se cambia el signo del resultado de la multiplicación. Finalmente se muestra por pantalla el resultado, se limpian los registros temporales y se finaliza el programa.

Resulta bastante simple llegar a este algoritmo, pues en realidad multiplicación en si es un ciclo de sumas, por lo que la única complicación como tal es traspasar el algoritmo a MIPS y comprobar el caso para los números negativos.

3.4 SOLUCIÓN PROBLEMA 3 PARTE B

Primero, se cargan los números que se encuentran en la sección .data del programa, y se muestran por consola. Se carga también el flotante 0,01, el cual será de ayuda para aquellas divisiones que den como resultado un flotante. Se comprueba si alguno o ambos de los números cargados son negativos, y si es así, se les cambia de signo, pero se deja registro a través de un 1 en los registros $t3 o $t4 si el primer o segundo número era negativo. Luego empieza el proceso de la división, el cual es bastante similar al proceso de la parte A, solo que se resta el primer numero con el segundo y se va aumentando un contador en 1, el cual corresponde a la parte entera de la división. Si la división no da un flotante, el proceso se acaba ahí. Pero si no, se continua, hasta que se obtenga la parte decimal, multiplicando el primer digito por 100 y haciendo la división nuevamente. Luego, se copia el número y se divide por 100. Se toma solo la parte entera y se multiplica por 100, para luego restar el resultado de la primera división del proceso de parte decimal con la última, así obteniendo la parte decimal, la cual se multiplica por 0,01 para transformarla a flotante y poder sumarla con la parte entera. Finalmente se comprueban los signos de los números y se cambia el signo del resultado si es necesario, para luego finalizar el programa.

Este algoritmo resulta un poco mas complicado de idear, pues si bien, la parte entera no tiene mucha diferencia con el proceso de multiplicación, se debe utilizar ese mismo algoritmo para obtener la parte decimal, mezclándolo también con el proceso de multiplicación. Se llegó a este algoritmo tras realizar distintas pruebas con números flotantes y multiplicaciones de estos.

3.5 SOLUCIÓN PROBLEMA 4 PARTE A – FUNCIÓN SENO

Se carga el número entero y se procede a seguir la siguiente formula:

$$\operatorname{sen} x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Figura N°1: Formula de la Serie de Taylor de sen(x)

Por lo tanto, se calcula el factorial del orden actual sumado a 1. Luego se eleva el número ingresado a la potencia del orden actual sumado a 1, utilizando la multiplicación creada en el problema 3 parte A. Posterior a eso se utiliza el algoritmo de la división creado en el problema 3 parte B para dividir el resultado de la potencia con el factorial. Una vez ya se tiene el resultado, se va comparado dependiendo del orden actual de si debe tener signo positivo o negativo. Cuando ya se tiene el signo listo, se suma el resultado del orden actual con el resultado total. Y dependiendo de si se ha llegado al orden máximo (Se recomienda dejar como orden máximo a 11) se repite el proceso si es que no, aumentando el

"inicialFactorial" en 2, para así continuar la secuencia. En el caso de los números negativos, se cambia el signo del resultado para reflejar el resultado correcto.

Cabe destacar de que se tuvieron que hacer cambios en las multiplicaciones y divisiones del problema 3 partes A y B, para que funcionaran con números flotantes y así no tener problemas con los ordenes mayores a 7, pues el factorial de números sobre 13 sobrepasa los 32 bits y no puede ser guardado en registros normales. Por la misma razón, se utilizaron registros $f para las subrutinas que solo trabajan con flotantes.

3.6 SOLUCIÓN PROBLEMA 4 PARTE A – FUNCIÓN LOGARITMO NATURAL

Se carga el número flotante y se procede a seguir la siguiente formula:

$$ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4}...$$

Figura N°2: Formula de la Serie de Taylor de ln(x+1), adaptada para ln(x)

Cabe destacar de que la serie de Taylor solo esta definida para valores entre 0 y 2, por lo que valores alejados a 2 ya comienzan a desviar bastante sobre el verdadero valor del logaritmo natural. Primero, se parte haciendo la resta de x-1. Para evitar trabajar con negativos (en caso de que el valor de x sea entre 0 y 1) se pasa a positivo. Luego se multiplica el número por 10 y se transforma a entero, esto es para no tener problemas con las subrutinas de la multiplicación y división creados en el problema 3. Se calcula la potencia del orden actual y se divide con el orden actual. Una vez se ha hecho el proceso, dependiendo del orden en el que se este trabajando, se divide por $10^{orden+1}$ (Por ejemplo, si es orden 2, se multiplica por 10 elevado a 3) esto para que el valor pueda retornar a su valor flotante original y tenga a lo menos 2 decimales, si es que el orden no es muy grande. Una vez ya se a transformado, se verifica el signo correspondiente al orden, y si el x quedo negativo tras la resta inicial de (x-1), se verifica si la potencia es impar para que quede negativo el resultado. Finalmente, se suma el resultado del orden actual con el resultado final y se repite el ciclo si no se ha llegado al orden máximo. **El programa funciona con orden 11 a lo máximo.**

La verdad es que fue bastante difícil idear el algoritmo, pero básicamente se casi todas las herramientas usadas en ejercicios anteriores (a excepción de recursión y stack). Si se elige orden 11, se debe tener cuidado de probar valores muy alejados de 1, pues demorará varios minutos en calcular las soluciones de los ordenes sobre 5. Al igual que la parte A, se adaptaron las subrutinas para trabajar con flotantes. Cabe destacar de que debido a que la división realizada en el problema 3 parte B retorna la gran mayoría de las veces dos decimales, en el proceso se pierden varios decimales, **por lo que algunos valores son cercanos al valor real, mientras que otros difieren bastante.**

3.7 SOLUCIÓN PROBLEMA 4 PARTE B

A través de una calculadora matemática online, se obtiene el valor real para los ordenes 1 hasta 11 de las series de Taylor de sin(x) y ln(x), evaluadas en tres números diferentes. Se procede a hacer la resta de estos valores con los resultados obtenidos a través de los programas de MIPS y se grafica la resta, la cual corresponde al error que hay entre el valor real y el valor calculado por MIPS.

Para mas detalles de las soluciones de todos los problemas, ver los resultados a continuación.

4. RESULTADOS

Los siguientes son algunos ejemplos mostrando los resultados obtenidos:

```
Por favor ingrese el primer entero: 289
Por favor ingrese el segundo entero: 285
El maximo es: 289
-- program is finished running --
```

```
Por favor ingrese el primer entero: -68
Por favor ingrese el segundo entero: 39
El maximo es: 39
-- program is finished running --
```

Figura N°3: Ejemplos de resultados del Problema

```
El primer numero es: 713
El segundo numero es: 589
El Maximo Comun Divisor entre ambos numeros es: 31
-- program is finished running --
```

```
El primer numero es: 12
El segundo numero es: -84
El Maximo Comun Divisor entre ambos numeros es: 12
-- program is finished running --
```

Figura N°4: Ejemplos de resultados del Problema 2

```
El primer numero es: -190
El segundo numero es: -3
El resultado de la multiplicacion entre ambos es: 570
-- program is finished running --
```

```
El primer numero es: 278
El segundo numero es: -24
El resultado de la multiplicacion entre ambos es: -6672
-- program is finished running --
```

Figura N°5: Ejemplos de resultados del Problema 3 Parte A

```
El primer numero es: -189
El segundo numero es: 17
El resultado de la division entre ambos es: -11.11
-- program is finished running --
```

```
El primer numero es: 1000
El segundo numero es: 5
El resultado de la division entre ambos es: 200
-- program is finished running --
```

Figura N°6: Ejemplos de resultados del Problema 3 Parte B

```
Porfavor ingrese un numero cercano a cero (Por ejemplo, 1, 2 o 3): 3
El seno de 3 con orden 1 es: -1.5
El seno de 3 con orden 2 es: 0.52
El seno de 3 con orden 3 es: 0.09000012
El seno de 3 con orden 4 es: 0.14000012
El seno de 3 con orden 5 es: 0.14000012
El seno de 3 con orden 6 es: 0.14000012
El seno de 3 con orden 7 es: 0.14000012
El seno de 3 con orden 8 es: 0.14000012
El seno de 3 con orden 9 es: 0.14000012
El seno de 3 con orden 10 es: 0.14000012
El seno de 3 con orden 11 es: 0.14000012

-- program is finished running --
```

```
Porfavor ingrese un numero entre 0.1 y 2: 0.8
El logaritmo natural de 0.8 con orden 1 es: -0.19999999
El logaritmo natural de 0.8 con orden 2 es: -0.21999998
El logaritmo natural de 0.8 con orden 3 es: -0.21739998
El logaritmo natural de 0.8 con orden 4 es: -0.21779999
El logaritmo natural de 0.8 con orden 5 es: -0.21773599
El logaritmo natural de 0.8 con orden 6 es: -0.21774659
El logaritmo natural de 0.8 con orden 7 es: -0.21774477
El logaritmo natural de 0.8 con orden 8 es: -0.21774508
El logaritmo natural de 0.8 con orden 9 es: -0.21774502
El logaritmo natural de 0.8 con orden 10 es: -0.21774504
El logaritmo natural de 0.8 con orden 11 es: -0.21774504

-- program is finished running --
```

Figura N°7: Ejemplo de resultados del Problema 4 Parte A

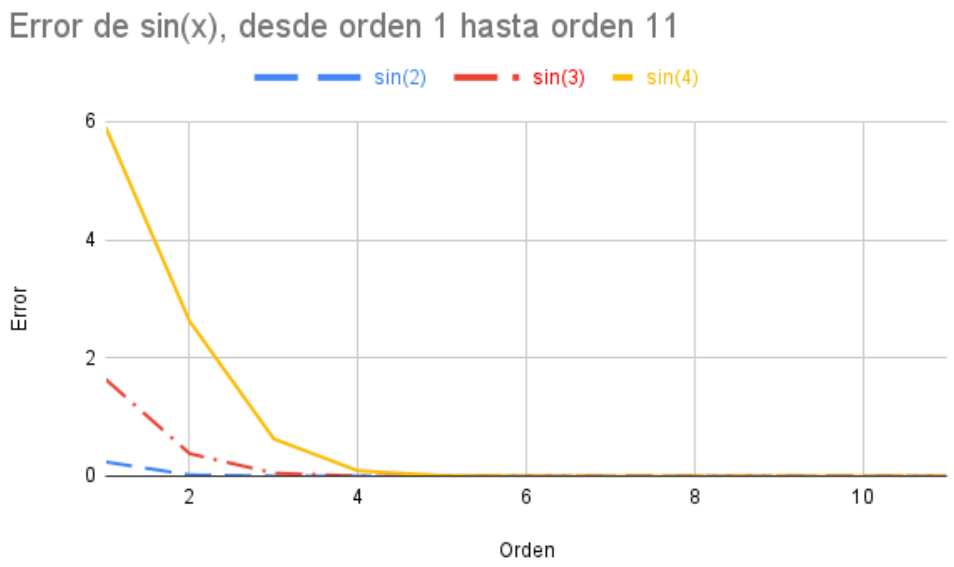

Gráfico N°1: Grafico de errores de la Serie de Taylor de Sin(x), probado con los números 2, 3 y 4.

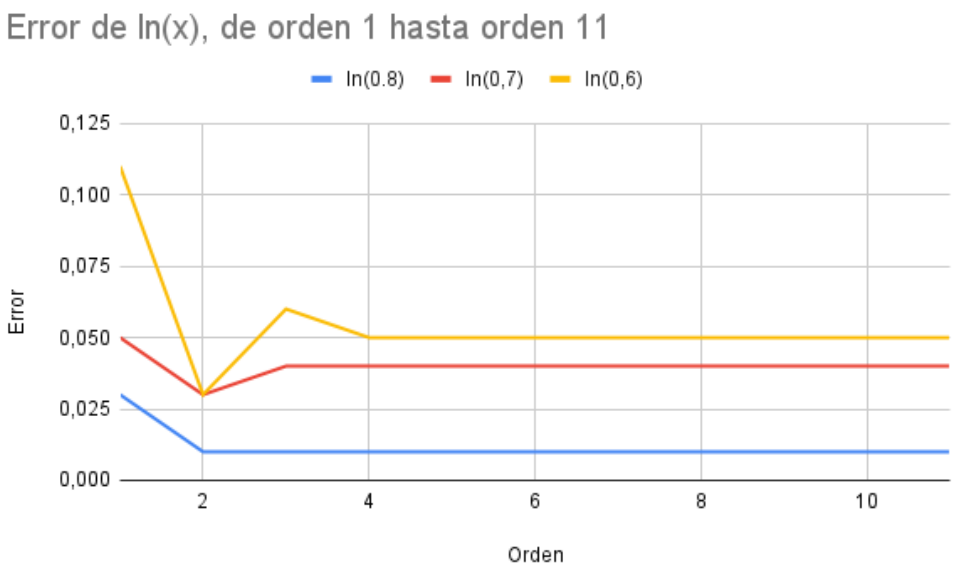

Gráfico N°2: Grafico de errores de la Serie de Taylor de Ln(x), evaluado con los números flotantes 0.6 , 0.7 , y 0.8.

5. CONCLUSIONES

En conclusión, se crearon programas para resolver problemas matemáticos utilizando el lenguaje ensamblador MIPS y los conceptos aprendidos en el laboratorio del curso de Arquitectura de Computadores. Los resultados fueron los esperados, con algunas pequeñas excepciones, ya que lamentablemente debido a la limitación de dos decimales de la división, no es posible crear un programa que refleje exactamente el comportamiento de la Serie de Taylor de la función Logaritmo Natural. Pero, de todas formas, se cumplieron los objetivos de la experiencia, pues se logró aprender sobre MIPS y utilizar estos nuevos conocimientos para llevar a cabo la solución a los problemas, utilizando conceptos como recursión, stack, instrucciones de salto, syscalls, entre otros. A pesar de que hubo algunas dificultades en el proceso de ideación, se logró de todas formas aprender sobre las herramientas adecuadas para resolver cada problema.

Se espera que en un futuro estos conocimientos sirvan para desarrollar soluciones a nuevos problemas, tanto en el curso de Arquitectura de Computadores como fuera de este.