**Organización de computadores**

**Laboratorio 3 – Acercándose al hardware:**

**Programación en lenguaje ensamblador**

Christofer Rodríguez

Profesor: Leo Medina

Ayudante: Ricardo Ruz

**Tabla de contenidos**

1. Introducción 3

2. Marco teórico 4

3. Desarrollo de la solución 5

3.1. Parte 1 5

3.2. Parte 2 6

3.3 Parte 3 7

4. Resultados 7

5. Conclusión 9

**1. INTRODUCCIÓN**

Para alcanzar un mayor conocimiento del lenguaje de ensamblador MIPS, se realizó una experiencia en la cuál fue necesario crear diferentes programas utilizando la herramienta MARS como IDE.

Los objetivos de la experiencia realizada son la utilización de MARS para escribir códigos en lenguaje ensamblador, comprender el uso de subrutinas, manejo del stack y la utilización de syscall para resolver problemas matemáticos.

Esto se llevará a cabo, primeramente, explicando los conceptos claves para el entendimiento de este informe en el marco teórico, se detallará el desarrollo de la solución para cada parte de la experiencia realizada, para finalmente analizar los resultados obtenidos y compararlos con los objetivos esperados.

**2. MARCO TEORICO**

* Lenguaje de máquina: Lenguaje de programación de bajo nivel (más lejos del simple entendimiento humano), este consiste en instrucciones para controlar directamente la CPU de un dispositivo.
* Lenguaje ensamblador: Lenguaje de programación de bajo nivel (difícil entendimiento humano), es el lenguaje más cercano al lenguaje de máquina.
* Subrutinas: Conjunto de instrucciones que realizan una tarea en especifica.
* Etiqueta: Nombre que se le otorga a una dirección de memoria en específico dentro del código.
* Double: Par de registros de “single precisión” (32 bits) utilizados para representar valores que no pueden ser representados en 32 bits, de esta manera se logra tener 64 bits.
* Potencia: Producto que resulta de multiplicar una cantidad o expresión por sí misma una o más veces.
* Expansiones de Taylor: Es una aproximación de funciones matemáticas mediante una serie de potencias, estas se calculan a partir de la derivada de la función en torno a cierto valor, es de interés para este caso las series de Maclaurin, que corresponden a la expansión de Taylor centradas en torno al 0.
* Sucesión de Fibonacci: Es una sucesión infinita de números naturales, en la cual se parte con un 0 y un 1, y cada elemento de la sucesión es obtenido al sumar los dos elementos anteriores a este.
* Recursión: técnica en la cual un proceso se utiliza a si mismo para su definición.
* Memoización: Técnica de optimización que consiste principalmente en almacenar los resultados “intermedios” en una memoria de fácil acceso para evitar que dichos procesos se repitan.
* Arreglo: Conjunto de datos que se encuentran ubicados de forma consecutiva.

**3. DESARROLLO DE LA SOLUCIÓN**

**3.1 Parte 1**

Para la primera parte fue necesario construir un programa capaz de calcular el valor de 3 funciones al evaluarlas con un numero entero no negativo mediante las expansiones de Taylor en torno al 0, en este caso se decidió hacer de un orden 7, esto quiere decir que la sumatoria va desde el 0 al 7. Como restricción a esta implementación, se requiere que esta se realice utilizando las operaciones de multiplicación, división y factorial creadas en el laboratorio anterior, esto quiere decir, sin utilizar las operaciones div o mul proporcionadas por el sistema.

Debido a que en el laboratorio anterior no se logró realizar el factorial para un número entero mayor a 12, se modificó la subrutina para que esta reciba como argumento un numero entero y tanto trabaje como retorne un double, de esta manera se podrá almacenar un numero el cual su representación binaria requiera más de 32 bits. A su vez se modifico la subrutina de división para que no dejara cada componente del número decimal (parte entera, primer decimal, segundo decimal) en registros separados, sino que trabaja con doubles y retorna el resultado en un double.

También se creó una nueva subrutina llamada potInt, esta se encarga de calcular la potencia del numero almacenado en el registro $a1 elevado al numero en el registro $a2. Primeramente, se verifican los casos bases, estos son exponente igual a 0 o a 1; en el caso de que sea igual a 0, se retornará un 1 y si el exponente es igual a 1, se retornará el valor de la base. El caso general consiste en utilizar el exponente como contador de un bucle, en cada ciclo de este se multiplica la base por el acumulado (inicialmente 1, neutro multiplicador) mediante la subrutina de multiplicación anteriormente explicada.

Además, para el correcto funcionamiento de las funciones a programar, se crearon diversas versiones de las operaciones, por ejemplo: multiplicación entre numero de 32 bits, multiplicación entre doubles, potencia para doubles, etc. Sumado a esto, también se incluyeron los casos dentro de las subrutinas de multiplicaciones y divisiones con números negativos.

En cuando al calculo de las funciones, se utilizaron las siguientes expresiones para cada uno:

Las tres funciones se construyeron de manera similar, dentro de un bucle primero se comienza por el lado izquierdo, calculado en numerado, el denominador y luego dividiendo, para después seguir con la siguiente expresión. Una vez se calculaba el valor obtenido en el ciclo, este era sumado con el acumulado almacenado en el registro $f22 (inicialmente 0).

**3.2 Parte 2**

En esta parte se pide calcular la sucesión de Fibonacci para un número indicado, esto se debe realizar de manera recursiva.

Para esto primero se establece el número del Fibonacci a calcular y se llama a la subrutina Fibonacci. Esta subrutina se encarga de calcular la sucesión de Fibonacci para el numero almacenado en el registro $a1, primero, ya que esta será una función recursiva en la que se realizarán “saltos” continuamente, se almacena en el stack la dirección de memoria desde la que se llamo a esta operación, el argumento que ingreso y el valor acumulado, este valor acumulado almacena la suma de todas las “subllamadas” realizadas por la llamada principal, será guardado en el registro $s0.

Luego se divide en dos casos principales, los casos bases y el caso general. El caso base consiste en verificar si se desea calcular el Fibonacci de 0 o 1, en el primer caso de retornará el valor 0 y en el segundo el valor 1, además devuelve la memoria utilizada por esta llamada en el stack y vuelve al lugar desde el que se llamo esta subrutina. Por otra parte, el caso general es para el Fibonacci de numero mayores a 1, en este caso es necesario calcular su valor con la expresión: por esto se almacena el argumento que entro a esta llamada menos 1 en el registro $a1 y se vuelve a llamar a la subrutina para que calcule el valor del Fibonacci de . Una vez calculado este valor, se suma el retorno con el valor acumulado en $s0, se entrega como argumento y se vuelve a llamar a la subrutina; nuevamente, una vez calculado este valor, se suma con el acumulado.

Para finalizar se recuperan los elementos almacenados en el stack, se restaura la memoria utilizada por estos y se retorna al lugar desde el que se llamo la subrutina. Cabe destacar que se dice se vuelve al lugar desde el que se hizo el salto, ya que la misma subrutina se llamará a si misma hasta alcanzar un caso base y comenzará a “retroceder” hasta llegar a la llamada principal, por esto es de suma importancia almacenar el argumento, el acumulado y la dirección de memoria desde donde se llamo en cada llamada, de esta manera se podrá realizar este “retroceso”.

**3.3 Parte 3**

En esta parte se requiere hacer uso de la memoización para mejorar el calculo de la sucesión de Fibonacci para cierto número. Para esto se decidió modificar el programa creada para la parte 2.

Haciendo uso de un “arreglo” para el cual la dirección de memoria desde donde comienza se almacenará en el registro $s7, dirección que será 0x10000000. En este arreglo se guardará el valor de cada Fibonacci calculado, en donde el índice del elemento almacenado corresponderá al número para el cual se calculó el Fibonacci (comenzando en cero), en un inicio se guardará el valor del Fibonacci de 0 y 1 en las dos primeras posiciones, los cuales son 0 y 1 respectivamente.

En cuanto a la subrutina, el único cambio que se realizo es que una vez se entra a esta función, se accede al valor almacenado en el arreglo en la posición mediante la instrucción lw, siendo el valor del Fibonacci a calcular y el 4 corresponde a la cantidad de bytes que utiliza cada valor; si este valor es diferente de 0, significa el ese Fibonacci ya ha sido calculado anteriormente y dicho valor corresponde al resultado, por lo que se retorna este valor. También se comprueba si el argumento es igual a cero, esto se realiza para no confundir el hecho de que el valor almacenado en el arreglo sea 0, significando que no se a calculado y el hecho de calcular el Fibonacci de 0.

Para el caso general, donde el valor obtenido del arreglo es igual a 0, se procederá al igual que en la versión original, se calculará mediante la expresión la única diferencia reside en que inicialmente se guarda en el stack la dirección de memoria donde se almacenara el resultado de esta llamada y una vez calculada la expresión anterior, se recupera esta dirección y se almacena el resultado, de esta manera podrá ser utilizado en otras llamadas sin ser calculado nuevamente.

**4. RESULTADOS**

Para comprobar el correcto funcionamiento de cada parte del laboratorio, se realizaron pruebas y sus resultados serán expuestos a continuación:

* Parte 1: Para la función del Coseno y Seno hiperbólico, se calcula correctamente la aproximación al valor real de estas funciones, el número difiere por el bajo orden de la serie y por el hecho de que los valores decimales solamente poseen 2 decimales.

Para el logaritmo natural, “teóricamente” debería funcionar, pero el valor entregado posee una gran diferencia con respecto al valor real, es surge por lo explicado anteriormente, de los 2 decimales, pero principalmente por el hecho de la forma en la que se implemente la multiplicación, ya que al utilizar el segundo argumento como contador y disminuirlo hasta llegar a un valor menor o igual a 1, en este caso en el que ambos números a multiplicar, tanto en la multiplicación principal, como en la potencia, se realizan multiplicaciones entre números menores a cero, lo que provoca que la subrutina solamente retorne el valor del primer argumento, cuando el verdadero resultado de una multiplicación de 2 numero menores a cero, debería ser menor al numero que se retorna, de esto proviene la gran diferencia entre el numero esperado y el obtenido.

Además, se analizó la eficiencia de cada función implementada comparando la cantidad de instrucciones escritas con la cantidad de instrucciones ejecutadas. El valor obtenido será expuesto a continuación:

Coseno:

* Instrucciones escritas: 238
* Instrucciones ejecutadas: 5351
* Eficiencia: 0,0445

Seno hiperbólico:

* Instrucciones escritas: 267
* Instrucciones ejecutadas: 6395
* Eficiencia: 0,0418

Coseno:

* Instrucciones escritas: 284
* Instrucciones ejecutadas: 73062
* Eficiencia: 0,0003

Cabe destacar que las pruebas se hicieron con x igual a 2 y la cantidad de instrucciones aumenta mientras el valor de x aumenta. Además, debido a que se realizan operaciones como multiplicación y potencia con operaciones básicas como la suma, en el caso de las 3 funciones, las implementaciones son ineficientes.

* Parte 2: Se implementó la subrutina capaz de calcular el Fibonacci de un número entero positivo, esta recibe, calcula y retorna el valor correctamente, para que este luego sea mostrado por consola.
* Parte 3: Se almacena la dirección del arreglo en el registro, se identifica correctamente cuando ya se ha calculado el Fibonacci deseado anteriormente y en el caso contrario, calcula el valor correctamente y los almacena el lugar correspondiente del arreglo para su posible utilización futura.

**5. CONCLUSIÓN**

A pesar de que no se logró alcanzar la totalidad de los objetivos propuestos, específicamente en llegar a la respuesta deseada a uno de estos problemas matemáticos, se considera que se pudo cumplir con los objetivos principales de entender el funcionamiento de las subrutinas, las llamadas de syscall, la utilización del stack y la recursividad . Como principal “ganancia” de esta experiencia podemos decir que se logró obtener una mayor familiaridad con el lenguaje ensamblador, el cuál será esencial para el desarrollo de futuros laboratorios.