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

Organización de Computadores

Laboratorio N°3

Alumno: Israel Arias
Sección: 0-L-1
Profesor(a): Leonel Medina

26 de Junio de 2021

Tabla de contenidos

1. Introducción

En la presente experiencia de laboratorio de la asignatura Organización de Computadores, se plantea una actividad en la cual se busca el aprendizaje de programar e implementar algoritmos en lenguaje ensamblador MIPS, para lo cual la actividad está dividida en tres partes, la primera busca la implementación de un algoritmo para aproximar funciones matemáticas, la segunda parte busca la implementación y aprendizaje de recursión en MIPS, finalmente la tercera parte busca implementar una técnica de "memoización". En el presente informe se detallarán los problemas abordados, su metodología de resolución, además se detallarán los objetivos específicos de la experiencia de laboratorio, se presentarán resultados para cada problema abordado y finalmente se efectuará una conclusión de acuerdo los resultados conseguidos.

1.1. Problemas

En la presente experiencia se plantean 3 problemas, los cuales serán detallados a continuación:

1.1.1. Parte 1: Aproximación de funciones matemáticas

En esta sección se plantea el siguiente problema: Escribir un programa que calcule una aproximación de las siguientes funciones, evaluadas en un número entero no negativo:

- 1) Coseno
- 2) Seno hiperbólico
- 3) Logaritmo natural

Para la aproximación se pide utilizar expansiones de Taylor en torno a 0, de orden 7 o superior y como restricción se solicita que el ingreso de datos debe ser solicitado al usuario y la implementación debe utilizar para calcular las multiplicaciones, divisiones y factorial, los procedimientos del laboratorio 2. Finalmente se pide cuantificar la eficiencia de la ejecución de instrucciones comparando el número de instrucciones efectivamente ejecutadas versus el número de instrucciones escritas.

1.1.2. Parte 2: Implementación en MIPS de una función recursiva

En esta sección se plantea el siguiente problema: Escribir un problema que calcule el elemento n-ésimo de la sucesión de Fibonacci según el esquema recursivo presentado, como condición se solicita no convertirlo a una solución iterativa.

1.1.3. Parte 3: "Memoización" en MIPS

En esta sección se plantea el siguiente problema: Modificar la implementación de la parte 2 para "memoizar" los resultados, o sea ir almacenando los resultados en un búfer para un posterior acceso más rápido. Además, se presenta una función modificada para implementar la memoización.

1.2. Soluciones

Las soluciones de los problemas presentados en la sección anterior se encuentran programados en lenguaje ensamblador MIPS, los códigos fuente de las soluciones pueden encontrarse dentro de esta misma entrega de laboratorio con los nombres: "parte1.asm", "parte2.asm" y "parte3.asm".

La solución para la parte 1 consistió en implementar la serie de Taylor de coseno, seno hiperbólico y logaritmo natural cada una en una subrutina, en la cual todos los cálculos se efectuaron con los procedimientos del laboratorio 2, la serie se expande en orden 7. El número por evaluar se le pide al usuario y finalmente entrega el resultado de las aproximaciones por pantalla para cada una de las funciones. La solución para la parte 2 consistió en implementar la recursión según la función de alto nivel entregada, para esto fue fundamental el uso del stack pointer y el uso de la instrucción jump and link para escapar y entrar a un punto de la función. La solución para la parte 3 consistió en modificar la implementación de la segunda parte, agregándole una dirección de memoria en la cual se iban escribiendo los resultados, lo que además permitió calcular mucho más rápido debido a que antes de calcular el termino n-1 y n-2 revisaba si ya se encontraban escritos en el arreglo, en caso de que estuvieran, solo los cargaba y no los calculaba. En la sección de "Desarrollo de la solución" se explicará la lógica y el desarrollo de la solución a cada uno de los problemas presentados con más detalle.

1.3. Objetivos

Los objetivos específicos para esta experiencia de laboratorio son:

- Usar MARS (un IDE para MIPS) para escribir, ensamblar y depurar programas MIPS.
- Escribir programas MIPS incluyendo instrucciones aritméticas, de salto y memoria.
- Comprender el uso de subrutinas en MIPS, incluyendo el manejo del stack.
- Realizar llamadas de sistema en MIPS mediante "syscall"
- Implementar algoritmos en MIPS para resolver problemas matemáticos

2. Marco Teórico

En esta sección se definirán distintos conceptos que se presentaran a lo largo del informe que se consideran relevantes para la comprensión del trabajo desarrollado:

- IDE: Integrated Development Environment, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- Lenguaje Ensamblador: Consiste en un conjunto de instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables.
- Instrucciones: Son el conjunto de funcionalidades que una unidad central de procesamiento puede entender y ejecutar
- MARS: Es un IDE que permite programar en MIPS
- MIPS: Son un conjunto de instrucciones que leen y escriben en memoria utilizando registros.
- Subrutina: Porción de código que realiza una operación en base a un conjunto de valores dados como parámetros de forma independiente al resto del programa y que puede ser invocado desde cualquier lugar del código, incluso desde dentro de ella misma.
- Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problema.
- Recursión: Es la forma en la cual se específica un proceso basado en su propia definición, significa que la solución depende de las soluciones de pequeñas instancias del mismo problema.
- Memoización: Es una técnica de optimización que se usa principalmente para acelerar los tiempos de cálculo, almacenando los resultados de la llamada a una subrutina en una memoria intermedia o búfer y devolviendo esos mismos valores cuando se llame de nuevo a la subrutina o función con los mismos parámetros de entrada.

3. Desarrollo de la solución

En esa sección se detallará el proceso y la lógica utilizada para resolver cada uno de los problemas planteados en esta experiencia de laboratorio.

3.1. Solución Parte 1

En el problema de la parte 1 se solicitó el aproximar a través de series de Taylor un número entero no negativo en las funciones coseno, seno hiperbólico y logaritmo natural, a continuación, se abordara de manera más detallada el cómo se implementó esta solución:

3.1.1. Aproximación Coseno

Para la implementación del cálculo del coseno, fue necesaria la implementación de la serie de Taylor en torno a 0 de la función coseno, la serie de Taylor para el coseno es:

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}\, x^{2n} \quad \text{para toda } x$$

Figura 1: Serie de Taylor en torno a 0 para la función Coseno

Como se solicitó que la implementación fuera de orden 7 o superior, la sumatoria se implementó en una subrutina la cual se ejecutaría desde 0 hasta 7 debido a que MIPS solo soporta hasta el factorial de 12 antes de un error por overflow, siendo controlada su salida por una instrucción beq, en cada ciclo de esta sumatoria se calcularía en primer lugar la parte de arriba de la división, o sea $(-1)^n$, esto se lograría gracias a la implementación de una subrutina llamada "calcularElevado", la cual calcula el resultado de elevar un número a otro haciendo uso de la operación multiplicar del laboratorio 2, luego este resultado quedaría almacenado en un registro, luego se calcularía x^{2n} haciendo uso de la subrutina "multiplicar" en primer lugar para calcular $2 * n$, luego de calculado $2 * n$ se haría el cálculo de x^{2n}, cuyo resultado queda almacenado en un registro, posterior a esto se efectúa la multiplicación de los resultados de $(-1)^n$ y x^{2n} haciendo uso de la subrutina "multiplicar", el resultado corresponde a la parte de arriba de la división. Luego se efectúa el cálculo del factorial $2n!$

que sera el divisor, para esto se hace uso de la subrutina “calcularFactorial”, con esto ya se tiene tanto el dividendo como el divisor para efectuar la división, por lo que se efectúa la división haciendo uso de la subrutina “dividir”, la subrutina dividir entrega el resultado de la división con dos decimales de precisión, luego para poder almacenar todos estos decimales como un solo número se convierte tanto la parte entera como los dos decimales a números flotantes en la subrutina “convertirDivisionFloat”, finalmente se acumula el resultado de la división como número flotante en el registro $f5, el cual ira almacenando todos los resultados de todas las iteraciones que tenga la sumatoria, finalmente se aumenta en uno el registro $t1 que hace de n en la sumatoria y se vuelve a ejecutar el ciclo. Esto hasta que se efectúen todas las sumatorias, una vez saliendo del ciclo se escribe el resultado de la aproximación disponible en $f5 por pantalla.

3.1.2. Aproximación Seno hiperbólico

Al igual que para la aproximación del coseno, fue necesaria la implementación de la serie de Taylor en torno a 0 de la función Seno hiperbólico, el cual es:

$$\operatorname{senh} x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} \quad \text{para toda } x$$

Figura 2: Serie de Taylor en torno a 0 para la función Seno hiperbólico

Y al igual que para el caso del coseno, esta sumatoria se implementó desde 0 hasta 6, debido a que MIPS solo soporta el factorial de 12 antes de un error por overflow, calculando en primer lugar x^{2n+1}, almacenando su resultado en un registro y luego calculando $(2n+1)!$, cuyo resultado también es almacenado en un registro, finalmente se efectúa la división, el resultado de la división se convierte a números flotantes y luego se va almacenando en el registro $f5, finalmente se aumenta el registro $t1 (n) en uno y se vuelve a ejecutar la iteración. Esto hasta que se efectúen todas las sumatorias, una vez se sale del ciclo se escribe el resultado de la aproximación disponible en $f5 por pantalla.

3.1.3. Aproximación Logaritmo natural

Al igual que para la aproximación de las anteriores funciones, fue necesaria la implementación de la serie de Taylor en torno a 0 de la función logaritmo natural, el cual es:

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\, x^n = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

Figura 3: Serie de Taylor en torno a 0 para la función logaritmo natural

Cabe notar que en este caso la función es para $ln(1+x)$, debido a que logaritmo natural es una función por tramos y la función $ln(1+x)$ es para casos positivos, que es lo que se pide en el enunciado. Debido a que se calcula la función evaluada $1+x$ en caso de que el usuario ingrese el número 2, se evaluara la función con el número 1 para conseguir la aproximación de $ln(2)$.
y al igual que anteriormente se implementó la serie a través de un ciclo, para este caso se implementó desde 1 hasta 52 para conseguir una aproximación más precisa. dentro del ciclo implementado en primer lugar se calcula $(-1)^{n+1}$ usando la subrutina "calcularElevado", luego se calcula x^n usando la misma subrutina, luego los resultados de ambos se multiplican usando la subrutina "multiplicar" y finalmente se divide este resultado con n usando la subrutina "dividir". Finalmente, al igual que en las otras dos funciones el resultado de la división se convierte a números flotantes y luego se va almacenando en el registro $f5, finalmente se aumenta el registro $t1 (n) en uno y se vuelve a ejecutar la iteración. Esto hasta que se efectúen todas las sumatorias, una vez se sale del ciclo se escribe el resultado de la aproximación disponible en $f5 por pantalla. La implementación se encuentra en la entrega de laboratorio con el nombre parte1.asm.

3.2. Solución Parte 2

Para la solución se implementó una subrutina con recursión natural, la subrutina emula la misma función dada, verificando en primer lugar los casos bases para luego llamarse a si misma recursivamente para calcular el Fibonacci de n-1 y n-2, almacenando la dirección del registro $ra y el valor de Fibonacci en el stack pointer $sp cuando corresponda a fin de

no perder ningún dato o alterar el flujo de la recursión. Cabe destacar que al llegar a un caso base se hace uso de la instrucción jr $ra para volver al punto donde se solicitó el cálculo y frenar la ejecución de las instrucciones siguientes. La implementación se encuentra en la entrega de laboratorio con el nombre parte2.asm.

3.3. Solución Parte 3

La solución de la parte 3 es la misma implementación de la segunda parte, con la diferencia que antes de entrar a la subrutina de cálculo de Fibonacci se inicializa una dirección de memoria (0x10010100) que cumplirá la labor de búfer o arreglo de memoria y almacenara los resultados de Fibonacci, la siguiente diferencia es que luego de comprobar los casos bases en caso de que n sea 0 o 1, se procederá a verificar si se encuentran los valores de n-1 o n-2 dentro del arreglo, para ello se calculara la dirección en la cual debería estar el número y si es un número distinto de cero (inicialmente el arreglo está lleno de ceros), significa que el número se encuentra escrito, el código que verifica si ya se encuentra el dato de Fibonacci calculado dentro del arreglo es el siguiente:

```
#Se verifica si esta el valor de fibonnaci n en el arreglo
   sll $t1, $a1, 2 #Se multiplica n por 4 para conseguir la posicion en el arreglo
   add $t1, $t1, $t5 #Se agrega a la direccion de memoria base del arreglo para
       conseguir la direccion exacta a revisar
   lw $v1, 0($t1) #Se carga el numero almacenado en el arreglo
   bne $v1, 0, recuperarValorArreglo #Si es distinto a cero significa que el
       valor si se encuentra dentro del arreglo
```

En caso de que efectivamente se encuentre el valor, dentro de la subrutina recuperarValorArreglo se hace uso de la instrucción jr $ra para volver a la dirección donde se solicitó el valor de Fibonacci, retornando así el valor del arreglo y frenando la ejecución recursiva para conseguir ese valor.

La otra modificación consiste en que al final de la rutina, al calcular el valor de un término de Fibonacci se escribe ese término en el arreglo, calculando su dirección de la misma forma presentada anteriormente.

4. Resultados

En esta sección se presentarán los resultados de cada problema.

4.1. Resultados Parte 1

Debido a la gran cantidad de números que se deben procesar, se recomienda probar el programa de la parte 1 con números pequeños, ya que para números más grandes toma una cantidad de tiempo muy grande para la ejecución, además debido a que MIPS se encuentra limitado, no es posible efectuar múltiples sumatorias debido a errores de overflow, por lo que es posible que para algunos casos la aproximación no sea similar al valor real. Cabe destacar que las series se encuentran en radianes, por lo que en caso de querer compararlas con sus valores originales debe ser en radianes. En todos los casos probados el programa cumple su función sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

4.2. Resultados Parte 2

El programa se ejecuta recursivamente solo llamándose a sí mismo hasta que consigue el valor de Fibonacci solicitado por el usuario. En todos los casos probados el programa cumple su función sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

4.3. Resultados Parte 3

Este programa es una mejora del programa de la parte 2 y almacena los resultados de Fibonacci en un búfer o dirección de memoria, lo cual resulta muy beneficioso, debido a que en vez de calcular resultados que ya se encuentran calculados, solo los recupera de este búfer de memoria, lo que resulta en un aumento del rendimiento al eliminar la redundancia de calcular cosas ya calculadas. A diferencia que el programa anterior, este muestra por pantalla todo el búfer de memoria con todos los valores de Fibonacci calculados hasta el número n ingresado por el usuario. En todos los casos probados el programa cumple su función sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

5. Conclusiones

Es posible concluir que esta experiencia de laboratorio fue satisfactoria, debido a que se cumplieron todos los objetivos planteados inicialmente:

- Se utilizo MARS para escribir, ensamblar y depurar cuatro programas MIPS.
- Se escribieron programas MIPS que incluían instrucciones aritméticas, de salto y uso de memoria.
- Se comprendió el uso de subrutinas en MIPS, siendo estas utilizadas en todos los programas de esta experiencia de laboratorio, además se hizo un manejo del stack en algunos programas para almacenar valores del registro $ra.
- Se realizaron llamadas del sistema mediante "syscall" que permitieron interactuar con el usuario o mostrar los resultados de una manera más comprensible por consola.
- Se implementaron algoritmos en MIPS que resolvían problemas matemáticos, como aproximar mediante series de Taylor las funciones de coseno, seno hiperbólico y logaritmo natural. además de calcular términos de Fibonacci y luego optimizar el algoritmo recursivo con memoización.

Para finalizar, en esta experiencia de laboratorio fue fundamental el correcto uso de subrutinas y con ello el correcto uso y almacenamiento de valores dentro del stack pointer, lo que permitió afianzar el conocimiento respecto al funcionamiento de ambas. Además, esta experiencia de laboratorio fue muy útil para interiorizarse en algoritmos recursivos en MIPS, además de aprender e implementar la técnica de memoización en forma de optimización.