Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Organización de computadores

Laboratorio 2 – Programación en lenguaje ensamblador

Christofer Rodríguez

Profesor: Leo Medina

Ayudante: Ricardo Ruz

Santiago – Chile
1-2021

Tabla de contenidos

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 de baja complejidad.

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.
- Registros: Parte del procesador que puede contener un patrón de bits, específicamente en MIPS, estos están compuestos de 32 bits.
- 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.
- Factorial: Producto de todos los enteros positivos desde 1 hasta el número del factorial a calcular.

3. DESARROLLO DE LA SOLUCIÓN

3.1 Parte 1

Para la primera parte era necesario implementar un programa que haciendo uso de subrutinas, mostrar el mayor valor de entre 2 números ingresados por el usuario. Para el ingreso de datos por parte del usuario, primero dentro del código se definieron mensajes para mostrar mediante la consola al momento de pedir los datos al usuario, luego se carga el valor 4 en el registro $v0, esto para indicar que la llamada de **syscall** mostrará un **string** en la consola, seguido se carga el mensaje dentro del registro $a0, registro utilizado para los argumentos de las "funciones"; una vez mostrado el mensaje por consola, el usuario debe ingresar uno de los valores y este será guardado en el registro $s0 para su futura utilización, esto se realiza cargando previamente el valor 5 dentro del registro $v0 para indicar que **syscall** leerá un entero. Este proceso se repite para el ingreso del segundo valor, con la diferencia de que se muestra el segundo mensaje y el valor ingresado es guardado en el registro $s1.

El calculo de mayor valor se realiza mediante la subrutina creada con el nombre **máximo**, esta subrutina recibe dos números almacenados en los registros utilizados para el paso de argumentos $a1 y $a2. Una vez dentro, se copian los valores en registros temporales para su utilización y mediante la operación **bge** se comprueba si el primero de estos valores ingresados es mayor que el segundo, en caso de que esta condición sea verdadera, se "saltará" a la etiqueta con el nombre **primerMayor**, la cual se encarga se copiar el primer valor (mayor) dentro del registro $v1 para ser retornado de esta subrutina. En caso de la condición comprobada en la operación **bge** no sea verdadera, significa que el segundo valor es mayor o igual al primero, por esto en cualquiera de estos dos casos se retorna el segundo valor mediante el registro $v1.

Una vez terminada la ejecución de la subrutina, se repite el proceso de cargar un mensaje en el registro $v0 para mostrar el texto indicando el resultado, luego se carga el valor 1 en $v0 para indicar que **syscall** mostrará un número y se muestra el número retornado por la subrutina, valor que representa en mayor numero de entre los 2 números ingresados por el usuario.

3.1 Parte 2

Para la segunda parte fue necesario implementar 3 programas que serán explicados a continuación:

Parte 2.a

Aquí fue necesario implementar una subrutina capaz de realizar una multiplicación entre dos números, sin hacer uso de las operaciones de multiplicación y otras operaciones similares. Ya que no se podía hacer uso de la operación para multiplicar normalmente, se decidió implementar una subrutina que haciendo uso de la definición de multiplicación llegara al resultado deseado, con esto nos referimos a una suma sucesiva.

La subrutina recibe ambos números mediante los registros de argumentos $a1 y $a2, y mediante la operación **beqz** se comprobaba si el segundo argumento de la operación era cero, en el caso de que esto sea cierto, se ejecutaban las instrucciones contenidas en la etiqueta **multiCero**, esta se encarga de retornar como resultado de la multiplicación el valor 0. Si el segundo argumento de la multiplicación no es cero, ambos valores eran copiados en registros temporales para su manipulación, luego se ejecutaba un ciclo que, utilizando el multiplicador como contador, suma el valor contenido en $t1 (registro donde se copió el valor del valor a multiplicar) consigo mismo, luego se resta uno a este contador y se vuelve al inicio de este ciclo mediante la operación de salto **j**. Al inicio de cada ciclo se comprueba si este contador es menor a 2, si es menor se salta a la etiqueta **exit** para retornar el valor contenido en $t1 (registro que contiene el resultado de la suma sucesiva) en el registro $v1; en caso de que el contador no sea menor a 2, se volverá a ejecutar este ciclo.

Terminada la ejecución de la subrutina, se muestra por consola un mensaje para mostrar el resultado, seguido del valor obtenido de esta subrutina; para finalmente terminar la ejecución del programa.

Parte 2.b

En esta parte se solicitó implementar un programa capaza de calcular el factorial de un número mediante la subrutina creada en la parte 2.a. Primero se establece el valor del factorial a calcular y se almacena en el registro $t3, junto con esto se guarda el valor 1 (neutro multiplicativo) en el registro $s0, registro que contendrá tanto el valor parcial como final del cálculo del factorial deseado.

Para calcular dicho factorial, se utilizará el valor de este factorial como contador y multiplicador, con esto nos referimos a que mediante un ciclo se multiplica con la subrutina construida previamente, el valor acumulado del registro $t3 (inicialmente 1) con este "contador" , una vez multiplicado, se le resta 1 a este contador y se vuelve al inicio del ciclo con la operación de salto **j**. Al inicio del ciclo se comprueba si el contador es menor a 2, de esta manera cuando se llega a 1 con el contador se retorna el valor contenido en el registro $t3, de esta manera al terminar de calcular el factorial se retorna la acumulación de las multiplicaciones sucesivas o el valor 1 si inicialmente se requería calcular el valor del factorial de 1 o 0.

Parte 2.c

Como ultimo programa se requería implementar un código capaz de realizar la división con al menos dos decimales entre 2 números enteros mediante subrutinas y sin utilizar operaciones de división, multiplicación y similares. Para esto se siguió inicialmente un enfoque parecido al de la multiplicación, siendo en este caso una resta sucesiva entre el dividendo y su divisor dentro de la subrutina llamada **división**, de esta manera se puede llegar a conocer cuantas veces "cabe" el divisor en el dividendo, y que al igual que las implementaciones anteriores, utiliza un ciclo para la realización de esta resta sucesiva.

Para comenzar se resta el valor del divisor al dividendo y su resultado es almacenado en un registro temporal diferente, luego se comprueba si el resultado de la resta es menor a cero, si este resultado no es menor a cero se aumenta en uno el contador almacenado en el registro $t4, después se comprueba si este resultado es igual a cero, si este es igual, se guarda el valor obtenido en el registro $s0 que representará la parte entera del resultado, además se guarda el valor cero en los registros $s1 y $s2 que representarán los dos primeros decimales del resultado; finalmente se retorna al cuerpo principal del código.

Si el resultado de la resta dentro del ciclo es mayor a cero, se volverá al inicio del ciclo con la operación de salto **j**, repitiéndose este proceso hasta que el resultado de la resta sea cero (caso explicado anteriormente) o menor a cero. En el caso de que la resta sea menor a cero, significa que la división que se está realizando es inexacta, por lo que se salta a la etiqueta **divInexacta**, al igual que en el primer caso, se almacena la parte entera de la división en el registro $s0, luego se guarda la dirección de memoria que apunta al lugar desde el que se ingresó a la subrutina **división** y el valor del divisor dentro del stack para no perderlos.

Una vez guardados los valores que no se quieren perder se hace uso de otra subrutina creada, esta se llama **subDivisionInexacta** para calcular el primer decimal del resultado, esta realiza una función similar a la de **división** con la diferencia que antes de realizar la resta sucesiva, multiplica el dividendo por 10 con la subrutina de la parte 2.a, y realiza esta resta sucesiva entre el resultado de la multiplicación del resto de la división original con 10 y el divisor original. Una vez el resultado de la resta es menor o igual a cero se retorna el valor del

contador de cuantas veces “cabe” el divisor en el dividendo y el resto de la división es asignado en el registro $a1 para ser utilizado en el cálculo del segundo decimal.

Después de volver de la primera llamada de **subDivisionInexacta** se almacena el valor obtenido en el registro $s1 que representa el primer decimal y se vuelve a repetir este mismo proceso para el segundo decimal, con la diferencia que se guarda el resultado en el registro $s2. Finalmente, el resultado de la división queda de la siguiente manera:” $s0”.” $s1” ” $s2” que equivale a “parte entera”.”primer decimal” ”segundo decimal”.

4. RESULTADOS

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

- Parte 1: Se implementó correctamente el proceso mediante una subrutina, los valores son ingresados por el usuario mediante la consola y se entrega la respuesta esperada.
- Parte 2.a: Se implementó la multiplicación de 2 números entero mediante una subrutina, pero dicha subrutina solamente es capaz de calcular el resultado correctamente cuando ambos números son enteros positivos o cero. Cuando uno de los argumentos es menor a cero, entrega como resultado el valor 1.
- Parte 2.b: Se implementó un código que calcula el factorial deseado correctamente para números menores a 13. Cuando el factorial a calcular es 13 o mayor, el programa finaliza con un error, esto provocado por el hecho de que el resultado no puede ser representado con 32 bits, haciendo que una de las sumas internas entregue un valor de que no se puede representar en 32 bits, generando dicho error.
- Parte2.c: Mediante el uso de subrutinas se logró calcular la división de 2 números enteros sin utilizar las operaciones indicadas, entregando la respuesta deseada para cada unos de los casos de prueba, a excepción de dividir por 0, en donde se entrega el valor 0 en vez de indicar que esta operación se indefine.

5. CONCLUSIÓN

A pesar de que no se logró alcanzar la totalidad de los objetivos propuestos, específicamente en llegar a la respuesta deseada a 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. 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.