UNIVERSIDAD
DE SANTIAGO
DE CHILE

Laboratorio N°2

Arquitectura de computadores.

Acercándose al Hardware: Programación en Lenguaje Ensamblador.

Nombre: Matias Silva

Sección: B-2

Fecha de Entrega: 07 de diciembre del 2021

UNIVERSIDAD DE SANTIAGO DE CHILE

Contenido

I. Introducción

A. Antecedentes

Los computadores modernos, al igual que las personas son capaces de interpretar y comprender un determinado lenguaje, este lenguaje consiste en series de instrucciones representadas en lenguaje binario (combinación de 0's y 1's). Dada la dificultad que presenta a una persona poder interpretar y comprender este tipo de lenguaje, se han ido generando distintas formas de poder abordar el problema de la interpretación y comprensión, es por esto, que se han generado distintos lenguajes de programación que permiten interactuar con el lenguaje de la maquina de una manera sencilla. Una de las primeras formas de interactuar con la maquina son los llamados lenguajes de bajo nivel, los cuales sirven para generar instrucciones entendibles para una persona, pero que pueden a su vez ejercer un control directo sobre la máquina, lo que conlleva a su vez una condicionalidad con el tipo de maquina que se esté utilizando. Para este laboratorio se utilizará una maquina o procesador MIPS el cual fue creado por John Hennessy y estudiantes de este en el año 1981 en la Universidad de Stanford, es considerado uno de los primeros procesadores RISC. La arquitectura de este procesador fue pensada con la idea de poder representar instrucciones de bajo nivel lo que conlleva a un lenguaje ensamblador de este. Por otra parte, es importante señalar que el procesador MIPS en su primera versión usaba 32 registros de 32 bits cada uno y, que presentaba un total de 111 instrucciones representadas en 32 bits.

Para el desarrollo de este laboratorio se utilizará el simulador MARS, el cual es capaz de trabajar como un procesador MIPS y permite escribir código de bajo nivel ensamblador que utiliza la maquina MIPS real.

B. Problema

Se presentan cuatro problemas diferentes los cuales deben ser resueltos utilizando el lenguaje ensamblador de bajo nivel de MIPS, el cual viene incluido en el simulador MARS, cada problema tiene un enfoque especifico los cuales se verán a continuación:

- Parte 1, Uso de syscall: Tiene como finalidad hacer uso de las llamadas de sistema "syscall". Para obtener esta solución se piden los siguientes pasos
    - Imprimir en la consola de MARS los mensajes de interacción con el usuario
    - Permitir que el usuario ingrese los números en tiempo de ejecución
    - Imprimir el resultado en la consola
    - Terminar el programa (syscall exit)

  Además, el cálculo del valor máximo debe ser realizado en una subrutina, utilizando registros adecuados para argumentos y salida de un procedimiento, la consola de entrada y salida debe seguir el siguiente formato:

```
Por favor ingrese el primer entero:  31
Por favor ingrese el segundo entero: 10
El maximo es: 31
-- program is finished running --
```

- Parte 2, Subrutinas para encontrar el máximo común divisor: Tiene como finalidad hacer uso de subrutinas y el stack para implementar una forma recursiva del algoritmo de Euclides. Para esto, se piden que los dos operandos enteros se encuentren escritos “en duro” en el mismo código y, deben estar claramente identificados para ser probados.

- Parte 3, Subrutinas para multiplicación y división de números: Esta parte se divide en dos subsecciones, pero ambas tienen como requisito el no utilizar instrucciones de multiplicación, división y desplazamiento.
    - Parte 3a: Tiene como finalidad calcular la multiplicación de dos enteros mediante la implementación de subrutinas (pueden ser iterativas o recursivas)
    - Parte 3b: Tiene como finalidad y, de manera similar a 3a, calcular la división de dos enteros mediante la implementación de subrutinas (que nuevamente pueden ser iterativas o recursivas)

- Parte 4, Subrutinas para multiplicación y división de números: Al igual que la parte anterior, esta se divide en dos subsecciones.
    - Parte 4a: Tiene como finalidad calcular una aproximación de las funciones seno y logaritmo natural. Para aproximar estas funciones se pide el uso de expansiones de Taylor en torno a 0. Se requiere utilizar para los cálculos de multiplicación y división correspondientes, las implementaciones resueltas en la parte 3a y 3b. Además, se debe pedir vía consola MARS el numero para el cual se quieren evaluar las funciones y, se debe mostrar en la misma consola los resultados de estas
    - Parte 4b: Tiene como finalidad cuantificar el error de la aproximación de la parte 4a en función del número de términos de la expansión de Taylor. Para ello se requiere construir un grafico que muestre el error para 3 números distintos utilizando la expansión de orden 1 hasta, al menos, orden 11.

C. Solución

Dado a que el laboratorio se divide en cuatro partes, se hablara de estas por separado a continuación.

- Parte 1: Para esta solución, se hace uso de las llamadas de sistema “syscall”, se utilizan para mostrar mensajes al usuario, interactuar con el mismo, entregar el resultado y finalizar el programa. Para revisar cual de los dos números es mayor se crean dos subrutinas, si el primer elemento ingresado es mayor o igual al segundo elemento, se entrega este como el mayor, en caso contrario, se entrega el segundo elemento ingresado como mayor.
- Parte 2: Para esta solución se entregan dos valores en duro y se hace uso del registro stack, para esto se asigna espacio al stack y se almacena la dirección de retorno en el mismo, luego, recursivamente y, con los elementos de este stack se van dividiendo y reemplazando los valores hasta que el resto sea 0, en este caso se devuelve a la posición de retorno (que esta guardada en el stack) y se muestra el resultado y termina el programa

- A continuación, un ejemplo:

Sean a=81 y b=18
Se divide a en b, lo que da como resultado 4 y resto 9
Ahora, a=18 y b=9
Se repite la división, lo que da como resultado 2 y resto 0
El resultado de la operación es 9

- Parte 3: Esta parte se divide en dos subproblemas, por lo que esta sección se dividirá en 2.
  - Parte 3a: Para esta solución se entregan dos datos en duro y, se ejecuta dependiendo de los signos de los datos iniciales como muestra en la figura:

A=2, B=4, C=0	A=-2, B=4, C=0	A=2, B=-4, C=0	A=-2, B=-4, C=0
B=B-1, C=C+A	B=B-1, C=C+A	A=A-1, C=C+B	Se niega A y B
Por lo tanto:	Por lo tanto	Por lo tanto:	Se hace el caso de positivos
B=3, C=2	B=3, C=-2	A=1, C=-4	
B=2, C=4	B=2, C=-4	A=2, C=-8	
B=1, C=6	B=1, C=-6		
B=0, C=8	B=0, C=-8		
Resultado: 8	Resultado: -8	Resultado: -8	Resultado: 8

    - Anotación: Si A o B son 0, entonces devuelve 0
  - Parte 3b: Para esta solución se entregan dos datos en duro, se generan copias de los datos iniciales y, luego se trabajan con los valores positivos de los datos. El proceso es parecido al mostrado en 3a, solamente que en este caso se hace la resta hasta el divisor sea mayor que el dividendo, en dicho caso se considera este ultimo dividendo como el resto y el resultado de la división como la parte entera. Una vez hecha la primera división, el resto se multiplica por 10 y se hace la división con este nuevo dividendo, este resultado es guardado en un arreglo, el cual almacenara la parte decimal del número. Se repite el proceso una vez mas para tener dos números guardados en el arreglo. Una vez listo, se obtiene cada elemento del arreglo, se transforma a flotante y se hace el proceso de multiplicación mostrado en 3a pero para flotantes, es decir, se multiplica el primer decimal por 0,1 y el segundo decimal por 0,2, una vez listo se suma la parte entera en forma de flotante con los dos decimales obtenidos. Si solamente una de las copias de los datos originales tiene signo negativo, entonces se niega el resultado final. En caso de que los valores originales fueran negativos, se mantiene el resultado final.

D. Objetivos

Los objetivos para este laboratorio son:

- Usar MARS (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 de stack
- Realizar llamadas de sistema en MIPS mediante “syscall”
- Implementar algoritmos en MIPS para resolver problemas matemáticos sencillos.

II. Marco teórico

- Máximo común divisor: El máximo común divisor entre dos enteros (ambos no ceros) es el entero mas grande que es capaz de dividir a ambos enteros. Es decir, sean dos enteros, a y b, el máximo común divisor c es aquel tal que a dividido en c entregue como resultado un numero entero y, b dividido en c entregue como resultado un numero entero.
- Algoritmo de Euclides: El algoritmo de Euclides es una de las formas de encontrar el máximo común divisor entre dos números positivos y no ceros. Este algoritmo funciona de la siguiente manera:
    - Se tienen como entrada dos números enteros, a y b
    - Si a es menor que b, se reemplaza a por b
    - Se divide a por b y se obtiene el resto, r. Si r=0, se entrega b como el máximo común divisor
    - Se reemplaza a por b y se reemplaza b por r y se repite el paso anterior.

  Este algoritmo tiene como salida, el máximo común divisor c para a y b.

- Syscall: Syscall son los servicios del sistema (System Services), son principalmente para entrada y salida de información. Estos funcionan en primer lugar, cargando el servicio que se quiera utilizar en $v0, luego se procede a cargar los argumentos ($a0, $a1, $a2 o $f12) y, por ultimo se hace la instrucción SYSCALL que entrega el resultado.
- Recursividad: Es el procedimiento en el cual una función se llama a si misma. Es utilizada para resolver problemas que contienen subproblemas del mismo tipo, pero mas pequeños. Para crear una función recursiva se requieren de dos condiciones necesarias. En primer lugar, se requiere de uno o mas casos bases, los cuales son capaces de terminar la recursión o entregar un resultado pendiente. En segundo lugar, se requiere de uno o mas casos recursivos que consisten en ejecutar la misma función nuevamente con un subproblema.
- Subrutinas: Consiste en un subalgoritmo presentado en un algoritmo mas grande, estos subalgoritmos resuelven problemas específicos necesarios para el resultado del algoritmo principal.
- Stack: Es una estructura de datos lineal la cual permite almacenar y utilizar variables que son almacenadas dentro de la misma.

III. Desarrollo de la solución

Dado a que el laboratorio se divide en partes, la siguiente sección también se encontrara divida en partes.

- Parte 1: Para este problema, primero se crearon los tipos de texto que se necesitaran mostrar por consola, estos son los textos para pedir al usuario que ingrese los números y, el texto que indica el número máximo.

  El programa empieza ejecutándose desde la subrutina MAIN, donde se imprime por pantalla el mensaje para ingresar el primer valor, el cual luego es almacenado en la dirección $v0 y, luego se obtiene una copia de esta en $t0, luego se repite el mismo proceso para el segundo dato y, se guarda en $t1.

  Se compara si $t0 es mayor o igual que $t1, en dicho caso se va a la subrutina MAYOR1 y imprime por pantalla el mensaje de máximo y el valor almacenado en $t0. Si $t1 es mayor, se va a la subrutina MAYOR2 y hace el mismo proceso, solo que esta vez imprime el valor almacenado en $t1

- Parte 2: Para este problema se tiene almacenado un mensaje, el cual sirve para entregar el resultado por consola.

  El programa empieza ejecutándose desde la subrutina MAIN, donde en primer lugar se entregan dos números en duro, luego estos elementos se pasan a las variables tipo $a, se llama a la subrutina EUCLIDES linkeando los valores.

  En EUCLIDES se asigna el espacio para el stack y se almacena la dirección de retorno en $ra, luego se llama a la subrutina RECURSIVIDAD en la cual se salta linkeando los elementos del stack a la subrutina DIVISION.

  En esta subrutina, se aplica el Algoritmo de Euclides con los elementos del stack $a0 y $a1, además se utilizan $a2 y $a3 para el nuevo dividendo y el resto (ver Marco Teórico para más información) y, por cada cambio de variables, se devuelve a RECURSIVIDAD para comprobar de que el resto sea distinto de 0, en caso de que lo sea, se vuelve a la subrutina DIVISION y, en caso contrario se salta a VOLVER.

  En VOLVER, se carga la dirección de retorno $ra para volver a MAIN y se libera espacio de stack, luego se devuelve a MAIN, donde se salta a la subrutina IMPRIMIR la cual imprime el resultado gracias a las instrucciones SYSCALL y vuelve a MAIN y, por ultimo se salta a END y finaliza el programa con la instrucción 10 de SYSCALL.

- Parte 3a: Para este problema se tiene almacenado un mensaje, el cual se muestra al final junto al resultado obtenido.

  El programa parte con la subrutina MAIN donde se cargan los datos en duro en $s0 y $s1, luego se salta a MULTIPLICACION. En esta subrutina se comprueba en primer si alguno de los dos números es 0, si ese es el caso, $s2 se vuelve 0 y se imprime dicho valor gracias a llamadas SYSCALL. En caso de que solo uno de los dos valores sea negativo se salta a MULT1, si $s1 es negativo y a MULT2 si $s0 es negativo.

Primero, veamos el caso de ambos positivos. En este caso, a $s1 se le resta 1 y, se crea una variable $s2 la cual es la suma de $s2 (que parte en 0) y $s0, si $s1 es distinto de 0, entonces se vuelve a llamar a la subrutina MULTIPLICACIÓN de manera iterativa. Esto se repite hasta que $s1 sea 0, en dicho caso se salta a IMPRIMIR, se imprime el valor guardado en $s2 y se finaliza el programa con la subrutina END, gracias a SYSCALL 10.

En caso de que alguno sea negativo, por facilidad tomaremos $s1 negativo, se comprueba que $s1 sea negativo y se salta a MULT1, en MULT1 se comprueba que $s0 sea negativo, si es negativo, se salta a MULT3 y se niegan ambos y se salta a MULTIPLICACIÓN. En caso de que $s0 no sea negativo se resta 1 a $s0 y se suma en $s2 el valor de $s2 y $s1, esto se repite hasta que $s0 sea 0, en dicho caso se salta a IMPRIMIR, donde se imprime el valor guardado en $s2 y se finaliza el programa con la subrutina END, gracias a SYSCALL 10. Lo anterior se repite si $s0 es el negativo, solamente que las operaciones aplicadas a $s1 son aplicadas a $s0 y, las operaciones aplicadas en $s0 son aplicadas a $s1, esto ocurre en la subrutina MULT2.

- Parte 3b: En primer lugar, se tiene espacio para un arreglo que será utilizado después, se tienen dos flotantes, flotante1 de valor 0.1 y flotante2 de valor 0.01, que serán ocupados posteriormente.

  Se inicia con la subrutina MAIN la cual carga en $s0 y $s1 los datos en duro, luego se hace una copia en $s5 y $s6 de $s0 y $s1 respectivamente, además, se carga en $f2 el valor de flotante1 y en $f4 el valor de flotante2, por último, se salta a DIVISION.

  En DIVISION, se comprueba si $s0 es negativo, en dicho caso se salta a TRANSFORMAR1 el cual niega el valor de $s0 y salta a DIVISION, se repite el mismo proceso si $s1 es negativo, pero esta vez utilizando TRANSFORMAR2. Luego, si $s0 es 0 se salta a END5 el cual muestra el valor entero y termina el programa, sino se comprueba si $s1 es mayor que $s0, en caso de que no ocurra se hace la resta $s0 = $s0 - $s1 y a $s2 se le suma 1, luego se salta a DIVISION. En caso de que $s1 sea mayor que $s0, se salta a RESTO donde se hace una copia de $s0 en $s3 y, se carga el valor 10 en $t0, luego se hace una copia de $s3 en $t1 y se salta a MULTIPLICACIÓN.

  En MULTIPLICACIÓN se comprueba que $t1 sea equivalente a cero, si esto no ocurre, a $t0 se le resta 1 y se hace $t2 = $t1+$t2, luego si $t0 no es 0 se salta a MULTIPLICACION nuevamente de manera iterativa. En caso de que $t0 sea 0, se salta a RESTO2. En esta subrutina se hace $s0 = $t2 y se carga un 0 en $t2, luego se salta a DIVISION2.

  Una vez en DIVISION2 se comprueba si $s1 es mayor que $s0, si no se cumple, se hace $s0 = $s0-$s1 y se le suma 1 a $s4 y se salta nuevamente a DIVISION2. En caso de que $s1 sea mayor que $s0 se salta a RESTO3. En esta parte se copia $s0 en $s3, se carga un 10 en $t0 y se copia $s3 en $t1, se comprueba que $t3 sea igual a 8, si no se cumple, se guarda en arreglo($t3) el valor de $s4, es decir, se guarda el primer decimal en la primera posición del arreglo, luego se le suma 4 a $t3 y se carga un 0 en $s4, por último, se repite el paso de MULTIPLICACIÓN. Una vez $t3 sea equivalente a 8 se salta a MULTFLOAT donde se carga un 4 en $t3 y se guarda en $t4 el primer elemento de arreglo, luego se transforma $t4 a flotante y se guarda en $f6, luego en $t5 se guarda el

segundo valor del arreglo, utilizando arreglo($t3) y se transforma a flotante y se guarda en $f8, por último, se salta a MULTFLOAT2

MULTFLOAT2 funciona de la misma manera que MULTIPLICACION solo que sirve para flotantes, por lo que se resta en $t4 el valor de 1 y en $f6 se suma $f2 (flotante1), una vez listo se salta a MULTFLOAT3.

Al igual que MULTFLOAT2 este hace la misma operación solo que en este caso a $t5 se le resta el valor de 1 y en $f8 se suma el valor de $f4 (flotante2), una vez listo se salta a END1.

En END1 se comprueba que $s5 sea menor que 0, si no lo es, se hace lo mismo para $s6, si no se cumple, se transforma $s2 en flotante (recordar que $s2 es la parte entera del resultado) y se guarda en $f16, luego se suma $f6 con $f8 y $f16 y se imprime el valor por pantalla gracias a SYSCALL y se finaliza el programa con otra llamada SYSCALL. Si $s5 es menor que 0 se salta a END2 donde se revisa si $s6 es menor que 0, en ese caso se salta a END4 donde se hace lo mismo que en END1. En END2 si $s6 no es menor que 0 se hace la suma en $f16 y se niega el valor, se muestra por pantalla y se termina el programa. Si en END1 $s5 no es menor que 0, pero, $s6 si lo es, se salta a END3, donde se repite el mismo proceso que en END2.

IV. Resultados

- Parte 1:
  - Entrada: 5 y 21
  - Salida: 21

```
Por favor ingrese el primer entero: 5
Por favor ingrese el segundo entero: 21
El maximo es: 21
-- program is finished running --
```

- Parte 2:
  - Entrada: 81 y 18
  - Salida: 9

```
li $s0, 81
li $s1, 18
```

```
El resultado es: 9
-- program is finished running --
```

- Parte 3a:
  - Entrada: -2 y 4
  - Salida: -8

```
li $s0, -2
li $s1, 4
```

```
El resultado es: -8
-- program is finished running --
```

- Parte 3b:
  - Entrada: -1 y 4
  - Salida: -0.25

```
li $s0, 1
li $s1, -4
```

```
-0.25
-- program is finished running --
```

V. Conclusión

Para concluir, podemos observar la suficiencia de objetivos.

Se utilizo de manera correcta MARS para escribir, ensamblar y depurar programas MIPS, por otro lado, los programas escritos incluyeron instrucciones aritméticas, de salto y memoria en los problemas que eran necesarios.

En segundo lugar, se pudo comprender el uso de subrutinas en MIPS, tanto de manera iterativa como de manera recursiva incluyendo el manejo de stack para esta ultima.

Se realizaron distintas llamadas en MIPS mediante SYSCALL, las cuales consistieron en mostrar texto por pantalla, pedir ingresar datos por pantalla, imprimir enteros y flotantes y, utilizar SYSCALL para finalizar los programas.

Por último, se implementaron distintos algoritmos en MIPS para resolver los problemas matemáticos sencillos como fue la implementación de encontrar un máximo entre dos números, encontrar el máximo común divisor gracias al algoritmo de Euclides y, la implementación de multiplicación y división de enteros.

En cuanto a las inconveniencias del laboratorio, podemos observar que la parte 4 del mismo no fue realizada, esto se debió al poco conocimiento en cuanto a las expansiones de Taylor y la dificultad que tenia al tener que utilizar las rutinas creadas en la parte 3.

Por otra parte, la parte 3b también tuvo bastantes inconveniencias, debido a que el manejo de enteros y flotantes a la hora de hacer divisiones y sumarlos fue complicado, dado que si bien, la parte de obtener el numero entero fue relativamente sencilla, la parte de obtener cada decimal fue complicado a la hora de no saber como trabajarlos ni como poder pasar un entero a flotante.