Laboratorio 2 - Subrutinas y uso de stack

Nicolás Aguilera (1.5 horas), Lucas Mesías (1.5 horas)
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
nicolas.aguilera.g@usach.cl

***Resumen*—El presente informe muestra la resolución de tres problemas mediante el uso de lenguaje de programación *assembler* haciendo uso del IDE MARS para MIPS, haciendo uso de subrutinas e instrucciones de salto. En primer lugar se introducen los problemas y su contexto, luego se presentan los antecedentes y el marco teórico dentro del cual se trabajará. A continuación se realiza una explicación de las soluciones y cómo se llegaron a ellas, para luego dar paso a los resultados de cada parte del laboratorio y finalizar con la conclusión del trabajo realizado.**

***Palabras claves*—MIPS, MARS, subrutinas y stack.**

I. Introducción

El siguiente informe tiene como objetivo la resolución de problemas en el lenguaje de programación *assembler* mediante el uso del IDE *MARS* para *MIPS*. Especificamente se busca, en primer lugar, la creación de un programa que permita la división entre dos números enteros sin utilizar instrucciones de multiplicación, división y desplazamiento, sino que utilizando otras operaciones aritméticas y subrutinas.

Los objetivos de este informe se presentan a continuación:

- Usar *MARS* para escribir, ensamblar y depurar programas, incluyendo instrucciones aritméticas y de acceso a memoria.
- Realizar llamadas de sistema en *MIPS* mediante uso de syscall empleando procedimientos.
- Implementar algoritmos en *MIPS* para resolver problemas matemáticos sencillos.

Las soluciones propuestas en este informe consisten primordialmente en el uso de subrutinas enlazadas mediante jal y $ra, el uso de stack para el problema de la *Distancia euclidiana*, para luego mediante el uso de *syscall* imprimir el resultado de la operación en la terminal con un formato legible, según sea el caso.

II. Antecedentes

MARS *(MIPS Assembler and Runtime Simulator)* es un IDE para MIPS que entrega herramientas para la escritura y depuración de códigos en lenguaje de programación *assembler* haciendo uso de un conjunto limitado de instrucciones[1], permitiendo manipular datos en los registros disponibles (a nivel de procesador) y los segmentos de datos (a nivel de memoria), ya sea cargando datos en los registros desde una memoria en particular o desde otro registro, por ejemplo, así como guardar cierto tipo de dato (entero, string, etc) en la memoria.

Existen diversos tipos de instrucciones dentro de *MIPS*[2], los cuales se presentan a continuacón:

- **Tipo R:** Instrucciones aritméticas.
- **Tipo I:** Instrucciones de transferencia de datos y de salto condicional.
- **Tipo J:** Salto incondicional.

Las instrucciones Tipo R son las que permiten sumar valores númericos entre registros o entre variables asignadas previamente. Las de Tipo I permiten, tal y como se mencionó anteriormente, cargar datos desde una dirección o guardarlos en una dirección específica. Finalmente las de Tipo J permiten por ejemplo, realizar ciclos *while* así como otro tipo de iteraciones.

II-A. Fórmulas

Las fórmulas utilizadas para el desarrollo de este informe son las ecuaciones para la *distancia euclidiana* entre dos puntos de un plano y la secuencia de Collatz:

$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2} \quad (1)$$

$$f(x) = \begin{cases} \frac{x}{2} & si \quad x \quad par \\ 3x+1 & si \quad x \quad impar \end{cases} \quad (2)$$

$$t_i = f(t_{i-1}) \quad (3)$$

donde la ecuación 2 es la función en la que se basa la secuencia de Collatz (ecuación 3) antes mencionada.

III. Materiales y métodos

III-A. Materiales

Para el desarrollo de esta actividad solamente se hizo uso del IDE *MARS* en su versión 4.5 sin herramientas o softwres adicionales.

III-B. Métodos

A continuación se describen los métodos utilizados para llegar a la solución de cada uno de los apartados del laboratorio.

III-B1. División: Dados dos números enteros x e y solicitados e ingresados por consola, se obtiene el resultado de su división mediante subrutinas iterando sumas y restas para guardar los valores del resto de la división, el dividendo y actualizar el divisor para la parte entera y fraccionaria.

III-B2. Distancia Euclidiana: Se obtienen por consola los vectores a y b, a continuación se utiliza la ecuación 1, donde se aproxima el cálculo de la raiz cuadrada haciendo uso del coprocesador 1 de MIPS y 7 iteraciones del método de Newton-Raphson usando una subrutina recursiva.

III-B3. Secuencia de Collatz: Utilizando las ecuaciones 2 y 3, se calcula la secuencia de Collatz para un t_0 arbitrario, almacenando la secuencia desde la dirección 0x100100A0 creando un comparador igual a 1 en el registro para verificar si la secuencia a terminado y cambiando la dirección de memoria cada vez que se guarda un valor.

IV. Resultados

Esta sección presenta los resultados obtenidos, siguiendo el mismo orden de las etapas descritas en la sección III.

IV-A. División

La solución consiste en la creación de varias subrutinas.

- **división:** Comprueba si la suma ha terminado, es decir, cuando el proceso de división ha calculado la parte entera y dos decimales de la parte fraccionaria mediante un contador global en $s0. Si no ha terminado suma el valor del divisor a un acumulador en $t1 y cuando este valor sea mayor que el dividendo, entonces ya no cabe más veces y se procede a guardar el valor.
- **guardar:** Se guarda la parte entera en la posición del puntero $s1 creado anteriormente y se mueve 4 bits, se reinicia el contador local de división en $t0 y se procede a multiplicar el resto de la división 10 veces mediante un proceso iterado de sumas para actualizar el dividendo y calcular la parte fraccionaria.
- **multiplicar:** Se suma 10 veces el valor del resto que se convertirá en el nuevo dividendo en $t2.

Se han creado otras sub rutinas como **absX** y **absY** que guardan el valor absoluto de los valores X e Y para la división, o las subrutinas de **casos** para comprobar cuál de los dos números es negativo/positivo y así imprimir el signo çuando corresponda.

En la Figura 1 se muestra un ejemplo de output del programa. A continuación se presenta la Tabla 1, el cual muestra los resultados obtenidos para 3 divisiones diferentes de dos enteros:

IV-B. Distancia Euclidiana

La primera subrutina que compone esta solución se compone de 2 partes:

- **Input:** Se piden por consola una de las coordenadas de ambos vectores y se almacenan en los registros $t1 y $t2.

N Prueba	X	Y	Esperado	Obtenido
Prueba 1	6	4	1.25	1.25
Prueba 2	-7	3	-2.33	-2.33
Prueba 3	1000	-12	-83.33	-83.33

Tabla I: Resultados obtenidos en tres pruebas del programa División implementado.

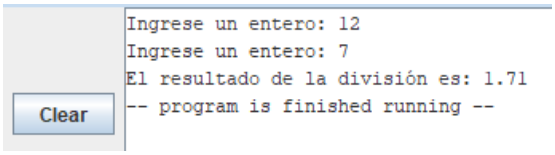

Figura 1: Output del programa Suma de dos vectores.

- **Calcular:** Se restan los valores recolectados y el resultado se eleva al cuadrado y se almacena/suma a $t0.

Al repetir este par 3 veces (una vez por cada coordenada [X, Y, Z]) se realizan todos los calculos interiores a la raiz de la ecuación 1, al terminar la subrutina se mueve el resultado calculado al registro $f10 del coprocesador 1, y es transformado de valor entero a punto flotante.
Ahora solo falta aproximar el valor de la raiz, donde haremos uso de la segunda subrutina del programa para hacer las 7 iteraciones del método de Newton-Raphson. Esta subrutina comienza preparando los valores necesarios para el cálculo, en el registro $f2 se almacena el valor inicial de la aproximación (1 en este caso), en $f4 un 2 constante y en $t0 la cantidad de iteraciones a realizar (7).
En cada iteración se realizan los calculos de numero flotante de doble precisión según la siguiente ecuación y se disminuye en 1 el iterador:

$$x_{n+1} = x_n - \frac{x_n^2 - X}{2x_n}$$

Donde X es el valor al que estamos aproximando la raiz, y está almacenado en $f10.
Cuando el iterador llega a 0 la subrutina se detiene y finalmente se imprime el valor aproximado en la consola.

N Prueba	s	v_2	Esperado	Obtenido
Prueba 1	$[1, 2, 3]$	$[4, 5, 6]$	5,196152422	5,196152423
Prueba 2	$[1, 1, 1]$	$[20, 20, 20]$	32, 908965343	32,936450175
Prueba 3	$[5, 20, 2]$	$[20, 5, 2]$	21, 213203435	21,213444817

Tabla II: Resultados obtenidos en tres pruebas del programa Distancia Euclidiana implementado.

IV-C. Secuencia de Collatz

El programa consta de varias subrutinas que se presentan a continuación:

- **main:** Solicita el t_0 arbitrario para comenzar la secuencia y *linkea* la subrutina **guardarVal**.
- **guardarVal:** utiliza el puntero en $s1 para guardar el valor de la secuencia partiendo de 0x100100A0 y se mueve 4 bits cada vez que se guarda un valor en $a1.

- **ciclo:** Nos lleva a la subrutina **collatz** y comprueba si la secuencia a terminado comparando el valor en $a1 con 1.
- **collatz:** Utiliza la ecuación 2 de *Fórmulas* para calcular el siguiente término (saltando a **collatzPar** si el valor de x es par) de la secuencia y así volver a la subrutina **ciclo** de donde fue "llamada". El valor obtenido es guardado en $v1 para posteriormente ser comparado con 1 y verificar si terminó el ciclo.

En este caso, la secuencia no es mostrada en consola debido a que el enunciado del problema no lo pide y por lo tanto no es necesario. De todas formas puede verse la secuencia tal y como fue pedida desde la memoria 0x100100A0, tal y como se muestra en la Figura 3. En la Tabla III se muestran los resultados de tres pruebas realizadas en el programa.

N Prueba	t_0	Obtenido
Prueba 1	5	5,16,8,4,2,1
Prueba 2	3	3,10,5,16,8,4,2,1
Prueba 3	4	4,2,1

Tabla III: Resultados obtenidos en tres pruebas del programa Secuencia de Collatz.

0x10010080	0	0	0	0	0	0
0x100100a0	5	16	8	4	2	1
0x100100c0	0	0	0	0	0	0

Figura 2: Output del programa *Secuencia de Collatz* para un t_0 arbitrario.

V. Conclusiones

Luego de realizar el laboratorio, se puede concluir que se ha cumplido con los objetivos planteados al comienzo de este informe. Se ha logrado escribir, ensamblar y depurar los programas solicitados en *MARS* mediante el uso de instrucciones aritméticas y de acceso a memoria, además de ejecutar procedimientos recursivos guardando los valores del contador de programa en el stack $sp.

Por otro lado, en cuanto a la solución propuesta para la división de números, se podría haber hecho un mejor uso de los registros, ya que al parecer no se respetó la convención debido al numeroso uso de registros. Sin embargo, la solución entregada es correcta y bastante completa.

Referencias

[1] P. Sanderson and K. Vollmar, "Mars: An education-oriented mips assembly language simulator," 2006, pp. 1–2.

[2] C. T. A. Hernández and R. Iglesias, "Arquitectura mips," in *Organización de Computadores*, 2010, pp. 1–2.