Laboratorio 1 - Manipulación de vectores en MIPS

Nicolás Aguilera (2.5 horas)
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
nicolas.aguilera.g@usach.cl

***Resumen*—El presente informe busca resolver un problema matemático mediante el uso de lenguaje de programación *assembler* haciendo uso del IDE MARS para MIPS. En primer lugar se introduce el problema y su contexto, luego se presentan los antecedentes y el marco teórico dentro del cual se trabajará. A continuación se realiza una explicación de la solución y cómo se llegó a ella, para luego dar paso a los resultados de cada parte del laboratorio y finalizar con la conclusión del trabajo realizado.**

***Palabras claves*—MIPS, MARS, assembler y registro.**

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 representar y realizar operaciones sobre vectores en $\mathbb{R}^3$ para el desarrollo de un módulo lógico-matemático de una nave no tripulada para explorar el oceano. Los vectores deben ser representados de la siguiente manera:

$$v = [x, y, z] \in \mathbb{Z}$$

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.

La solución consiste primordialmente en la asignación de datos almacenados en el segmento de datos (direcciones de memoria) a los registros temporales (como $\$t0$, por ejemplo) y aplicar operaciones aritméticas para luego, mediante el uso de *syscall* imprimir el resultado de la operación en la terminal con un formato legible.

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 suma, ponderación por escalar y producto punto para vectores en $\mathbb{R}^3$. Sean v_1 y v_2 vectores y $c \in \mathbb{R}$:

$$v_1 + v_2 = [v_{1x} + v_{2x}, v_{1y} + v_{2y}, v_{1z} + v_{2z}] \quad (1)$$

$$c \cdot v_1 = [c \cdot v_{1x}, c \cdot v_{1y}, c \cdot v_{1z}] \quad (2)$$

$$v_1 \cdot v_2 = v_{1x} \cdot v_{2x} + v_{1y} \cdot v_{2y} + v_{1z} \cdot v_{2z} \quad (3)$$

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. Suma: Dado que los vectores v_1 y v_2 están almacenados en el segmento de datos, la solución se obtuvo mediante la carga y manipulación de los datos en los registros temporales, es decir, la suma de los registros para cada coordenada correspondiente y luego la impresión del resultado en la consola.

III-B2. Multiplicación por escalar: En este caso, dado que la constante s (la cual ponderará cada coordenada del vector), debe ser pedida por terminal y el vector v se encuentra en el segmento de datos, el método de solución consiste en primer lugar en cargar la constante ingresada a algún registro temporal, al igual que las coordenadas del vector v para luego realizar las multiplicaciones correspondientes entre a constante s y las coordendas para finalmente imprimir el resultado en la terminal.

III-B3. Producto punto: Similar al inciso anterior, esta vez se deben ingresar las tres coordenadas del vector v_1 por consola y el v_2 se asume en el segmento de datos, por lo que el método de solución consiste en cargar cada una de las coordenadas de v_1 en los registros temporales y luego multiplicar cada una de estas con la coordenada correspondiente para finalmente sumar estas tres multiplicaciones e imprimir el resultado en la consola.

IV. Resultados

Esta sección presenta los resultados obtenidos, siguiendo el mismo orden de las etapas descritas en la sección III. Se debe acompañar de una explicación y un análisis en profundidad. Está permitido el uso de tablas y gráficos.

IV-A. Suma

En este caso la solución consiste en la reservación de espacio en el segmento de datos para las coordenadas del vector v_1 mediante el uso de *.space*, reservando exactamente el espacio equivalente a tres enteros, es decir 12 bits. Luego, para los espacios restantes, antes de los reservados para el vector v_2 se asignan strings que ayudarán a dar formato a la impresión del resultado en consola, usando *.asciiz*. Luego, al igual que con v_1, reservamos los espacios para v_2 y luego asignamos espacio a un último string, también para dar formato a la impresión del resultado. Una vez asignadas las variables y espacios de memoria para vectores, se procede a cargar cada valor entero encontrado en las direcciones de memoria de M[0x1001000], M[0x1001004] y M[0x1001008] en los registros $t0, $t1 y $t2, respectivamente. De igual forma con v_2 se cargan las direcciones de memoria de M[0x1001020], M[0x1001024] y M[0x1001028] en los registros $t3, $t4 y $t5. Luego, se procede a sumar los registros y guardarlos de la siguiente manera:

$$R[\$t0] = R[\$t0] + R[\$t3]$$
$$R[\$t1] = R[\$t1] + R[\$t4]$$
$$R[\$t2] = R[\$t2] + R[\$t5]$$

Finalmente, se prepara al procesador mediante la instrucción *li* para imprimir el mensaje que mostrará el resultado, usando *syscall*. De igual forma con los valores enteros, preparamos la impresión mediante *li* y se ejecuta mediante *syscall*, entregando el output mostrado en la Figura 1. A continuación se presenta la Tabla 1, el cual muestra los resultados obtenidos para 3 sumas diferentes de vectores:

N Prueba	v_1	v_2	Esperado	Obtenido
Prueba 1	$[1, 2, 3]$	$[2, 0, 1]$	$[3, 2, 4]$	$[3, 2, 4]$
Prueba 2	$[1, 0, -2]$	$[0, 0, 0]$	$[1, 0, -2]$	$[1, 0, -2]$
Prueba 3	$[-1, -2, -3]$	$[1, 2, 3]$	$[0, 0, 0]$	$[0, 0, 0]$

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

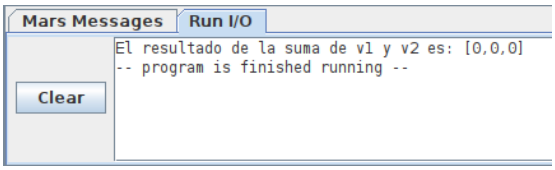

Figura 1: Output del programa Suma de dos vectores.

IV-B. Multiplicación por escalar

Al igual que en la suma, la solución consistió en la asignación de espacio para punteros de tipo *string* en el segmento de datos, para dar formato a la impresión del resultado en consola mediante *.asciiz*, y cargar los enteros encontrados en las direcciones de memoria M[0x1001040], M[0x1001044] y M[0x1001048] del vector v_2 a los registros temporales $t0, $t1 y $t2, respectivamente. Luego se procede a pedir el valor del escalar s mediante la instrucción *li* y se mueve el resultado al registro $t3. A continuación se procede a realizar la multiplicación entre escalar y cada coordenada, guardando los valores resultantes de la siguiente manera:

$$R[\$t0] = R[\$t0] \cdot R[\$t3]$$
$$R[\$t1] = R[\$t1] \cdot R[\$t3]$$
$$R[\$t2] = R[\$t2] \cdot R[\$t3]$$

Al igual que en Suma se prepara al procesador para imprimir el resultado con formato mediante *li* para los strings asignados y los resultados de la multiplicación. Un ejemplo de resultado puede verse en la Figura 2. A continuación se presenta la Tabla II, la cual muestra los resultados obtenidos en tres pruebas realizadas en el programa.

N Prueba	s	v_2	Esperado	Obtenido
Prueba 1	2	$[2, 0, 1]$	$[4, 0, 2]$	$[4, 0, 2]$
Prueba 2	-1	$[1, 2, 3]$	$[-1, -2, -3]$	$[-1, -2, -3]$
Prueba 3	0	$[1, 2, 3]$	$[0, 0, 0]$	$[0, 0, 0]$

Tabla II: Resultados obtenidos en tres pruebas del programa Multiplicación por escalar implementado.

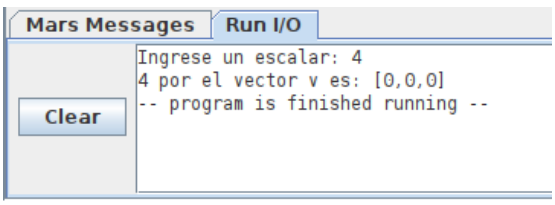

Figura 2: Output del programa Multiplicación por escalar entre un escalar s y un vector v.

N Prueba	v_1	v_2	Esperado	Obtenido
Prueba 1	$[1, 1, 1]$	$[1, 1, 1]$	3	3
Prueba 2	$[1, 0, -2]$	$[0, 0, 0]$	0	0
Prueba 3	$[-1, -2, -3]$	$[1, 2, 3]$	-14	-14

Tabla III: Resultados obtenidos en tres pruebas del programa Producto punto.

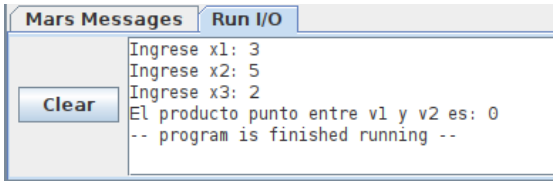

Figura 3: Output del programa *Producto punto* entre dos vectores.

IV-C. Producto punto

En este caso el vector v_1 debía pedirse por consola y el vector v_2 se asumía en el segmento de datos. Similar a la Multiplicación por escalar, se asginaron strings en el segmento de datos mediante *.asciiz* para darle formato a la impresión del resultado en consola. Luego, las coordendas del vector v_2 se cargaron desde el segmento de datos mediante *la* a los registros \$t0, \$t1 y \$t2. Por otro lado, los valores de las coordenadas de v_1 son pedidos por consola mediante la instrucción *li, $v0, 5* y luego, mediante la instrucción *move* se mueven a los registros temporales \$t3, \$t4 y \$t5, respectivamente. A continuación se multiplican los valores de las coordenadas de la siguiente manera:

$$R[\$t0] = R[\$t0] \cdot R[\$t3]$$
$$R[\$t1] = R[\$t1] \cdot R[\$t4]$$
$$R[\$t2] = R[\$t2] \cdot R[\$t5]$$

Luego se suman estos valores y se guardan en \$t0. Al igual que en los dos programas anteriores, se procede a imprimir el resultado con formato legible en pantalla mediante la instrucción *li* y *syscall*. Un ejemplo de resultado puede verse en la Figura 3. En la Tabla III se muestran los resultados de tres pruebas realizadas en el programa.

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 mediante *syscall*.

Debido a la naturaleza del problema, no fue necesario escatimar en uso de memoria a la hora de cargar valores desde el segmente de datos, sin embargo, podría haberse hecho un uso más acotado de los registros temporales, como en el caso de la solución propuesta para el programa *Suma de dos vectores* en donde se cargaron los valores de las direcciones de memoria en un total de 6 registros, pudiendo haber cargado de dos en dos y realizar la suma de forma inmediata, y así utilizar menos registros. Esto aplica también para los otros programas. Otra posible solución podría haber sido el uso de punteros y un contador para moverse por los espacios de memoria asignado para las coordendas de los vectores, de esta forma, si se tuviera un vector más largo, sería más práctico moverse a través de este mediante un çicloçon las instrucciones *beq* y *j*. A pesar de esto, los programas no presentan problemas y cumplen con los requisitos, entregando los resultados pedidos.

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.