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

Organización de Computadores

Laboratorio 1: MIPS

Nombre: Aylin Castillo

Sección: L-1

Profesor: Viktor Tapia

Ayudante: Nicolas Henríquez

Introducción

En el presente informe se documentará los distintos resultados obtenidos al realizar las actividades del laboratorio 1 de la asignatura Organización de Computadores, el cual tiene como objetivos el poder predecir el funcionamiento de códigos MIPS y poder pasar de código C a MIPS de manera correcta, además de aprender a usar la aplicación MARS (simulador de MIPS).

En específico, el problema que se busca abordar en esta experiencia se divide en dos partes, una es la predicción del funcionamiento de códigos MIPS y la otra de la traducción de estos con uso de ciclos y arreglos. Esto se buscará solucionar haciendo uso de los conocimientos entregados por la asignatura hasta el momento y además la documentación oficial de MIPS.

Marco Teórico

En el informe se hará uso de distintos conceptos los cuales serán descritos a continuación:

- **Arquitectura MIPS:** sus siglas traducidas al español significan “microprocesador sin bloqueos en las etapas de segmentación (De Nihil, 2019).
- **Lenguaje ensamblador:** formato legible por humanos de instrucciones (Tapia, 2024).
- **Lenguaje de máquina:** formato legible por ordenador (1 y 0) (Tapia, 2024).
- **MARS:** programa hecho con Java que puede ensamblar y simular la ejecución de un programa realizado en el lenguaje ensamblador MIPS (*MARS MIPS Simulator - Missouri State University*, s. f.).
- **Registros:** pequeños montones de memoria que se encuentran en la CPU que son de acceso muy rápido (De Nihil, 2019).
- **Arreglo (Array):** colección ordenada de datos a los cuales se puede acceder a través de una dirección de memoria (Sanchez, s. f.).

Desarrollo de la Solución

Para comenzar se respondieron las siguientes preguntas con respecto a MIPS:

- ¿Cómo se ensambla un programa MIPS?
- ¿Cómo se ejecuta un programa en MIPS?
- ¿Cómo detendrías la ejecución en cierta línea que no es necesariamente la última?
- ¿Dónde encontrarías el valor actualmente almacenado en el registro $s0?

Haciendo uso de la información obtenida por los vídeos de Youtube dados en el enunciado y logrando un mejor entendimiento de lo básico del lenguaje ensamblador.

Después, haciendo uso de de lo aprendido en clases sobre las instrucciones de MIPS

e información adicional de distintos artículos, se prosiguió con el desarrollo de la sección de predicción de instrucciones, la cual correspondía a señalar que es lo que realizaba cada línea de código y que valor es el que quedaba después de finalizado el programa.

Para finalizar se realizó la traducción de tres códigos escritos en C a lenguaje ensamblador con el uso de ciclos y arreglos con sus respectivas direcciones de memoria, los cuales fueron probados en el simulador MARS y se probaron con distintas entradas.

Resultados

Después de terminado el desarrollo se obtuvieron los siguientes resultados:

1. **Preguntas:**
   P1: ¿Cómo se ensambla un programa MIPS?
   R1: Primero hay que escribir el seudocódigo en formato MIPS en el apartado EDIT para luego convertirlo en instrucciones hexadecimales al completar el proceso de ensamblaje.

   P2: ¿Cómo se ejecuta un programa en MIPS?
   R2: Después de ensamblar con éxito el programa y haber reseteado (registros en cero), se hace click en el botón “run” para ejecutar el programa en MIPS y la consola mostrará su resultado final. En la ventana “execute” se podrá ver a más a detalle como quedaron los registros.

   P3: ¿Cómo detendrías la ejecución en cierta línea que no es necesariamente la última?
   R3: Con un breakpoint, los cuales se pueden fijar en la pestaña de ejecución (execute)

   P4: ¿Dónde encontrarías el valor actualmente almacenado en el registro $s0?
   R4: En la pestaña execute, al costado derecho donde están los registros y sus respectivos datos.

2. **Predicción de instrucciones:**
   2.1 addi y add suman los registros $t1 y $t0 y mul los multiplica y guarda el valor en $t2, dando así lo indicado en la figura 1.

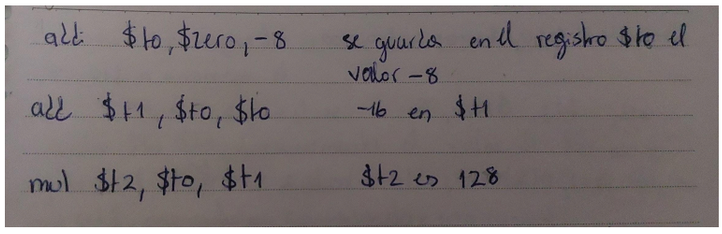

Figura 1: Respuesta ejercicio 1.

2.2 El código 2 se divide en 2.a y 2.b debido a que cambia el valor inicial de $t2 junto a sus respectivos valores finales.

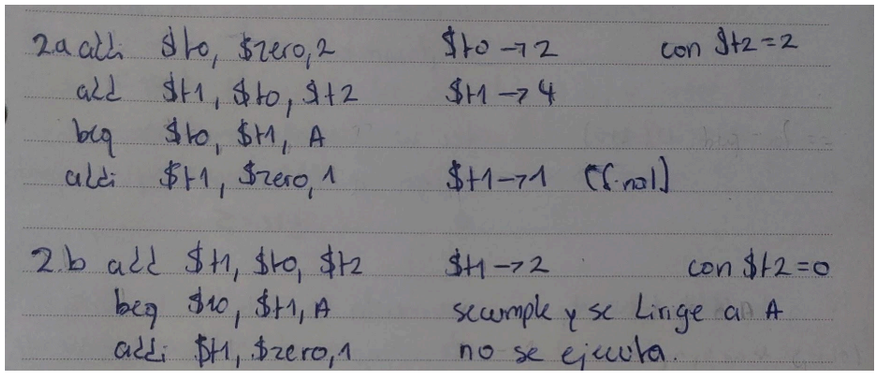

Figura 2: Respuesta ejercicio 2.

2.2.1 código 2.1 del enunciado junto a sus respectivos valores finales.

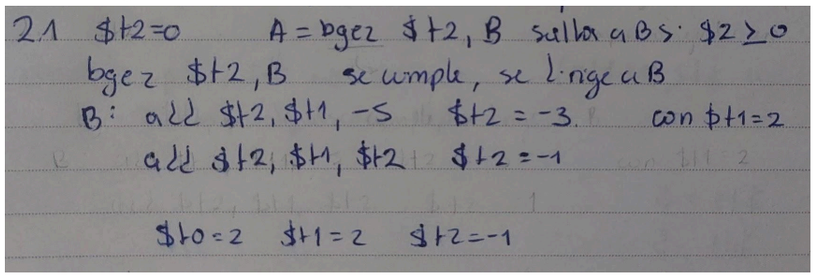

Figura 3: Respuesta ejercicio 2.1.

2.3 código 3 junto a sus respectivos valores finales.

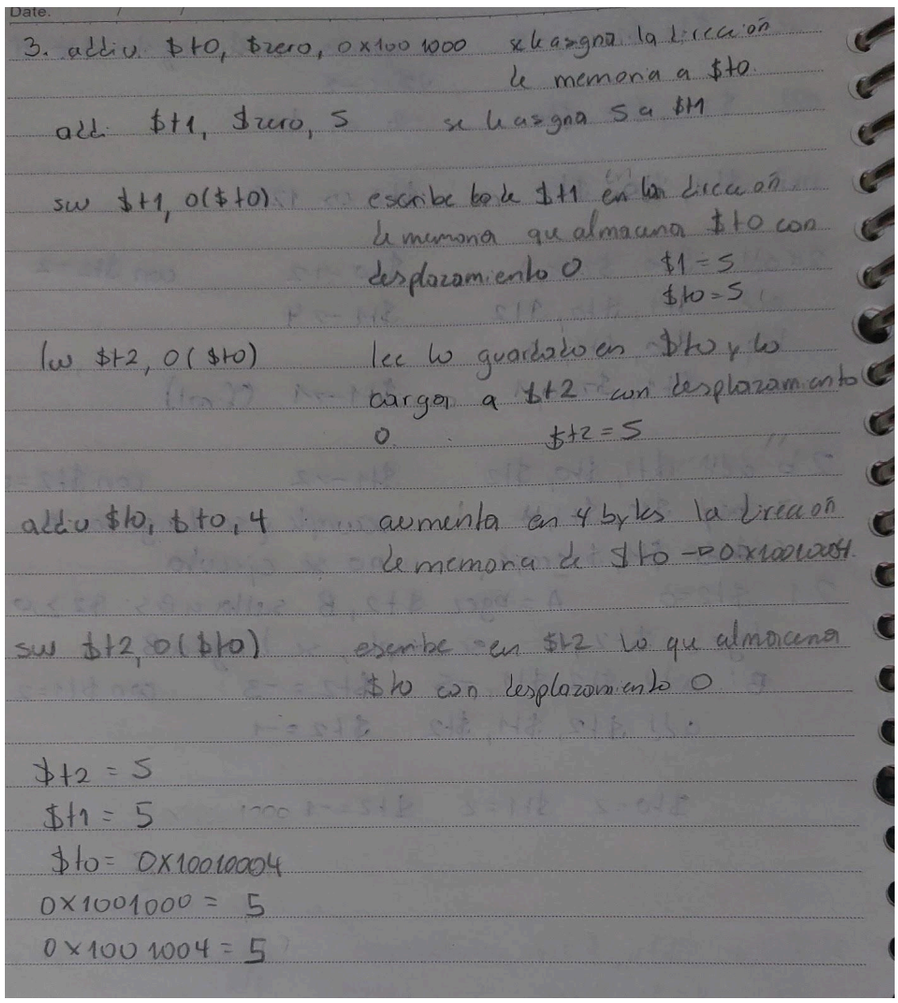

Figura 4: Respuesta ejercicio 3.

3. Traducción de C a MIPS.

En esta sección se mostrarán fragmentos del código traducido y junto a ellos tablas con los distintos resultados obtenidos en MIPS.

3.1 En el siguiente código se realizó la traducción de un ciclo while a MIPS haciendo uso de beq y add. En la figura se puede ver con más detalle su comportamiento.

```
addi $s0, $zero, 0 #se asume que es a
addi $s1, $zero, 1 #se asume que es z
addi $t0, $zero, 10


while:
        beq $s1, $t0, done #si z = 10, salta a done
        add $s0, $s1, $s0 #sino a = a + z
        add $s1, $s1, 1 # z = z + 1
        j while #salto a while para continuar con el ciclo

done:
```

Figura 5: Respuesta ejercicio 1.

El código anterior se evaluó con los siguientes valores:

Variable	Registro	Valor inicial	Valor final

a	$s0	0	45
z	$s1	1	10

Tabla 1: Distintos valores obtenidos en ejercicio 1.

3.2 En el siguiente código se realizó la traducción de C a MIPS de D[a] = b+a dependiendo de un ciclo while haciendo uso de slt, beq y sw, ya que se debía interactuar con la dirección de memoria del arreglo de prueba. En la figura se puede ver con más detalle su comportamiento.

```
.data
D: .word 1 2 3 4 5 6 7 8 9 10 #arreglo de prueba
.text
addi $s0, $zero, 0 #se asume que es a
addi $s1, $zero, 4 #se asume que es b
addi $s2, $zero, 0 #index, dirección base de memoria de D
addi $t0, $zero, 10

loop:
        slt $t1, $s0, $t0 #ve si a < 10
        beq $t1, 0, done #si a no es menor a 10, se va a done
        add $t3, $s0, $s1 #si a es menor a 10, hace a + b
        sw $t3, D($s2) #guarda la suma en D[a]
        #imprime el arreglo a medida que avanza
        li $v0, 1
        move $a0, $t3
        syscall
        #sección de avance en el arreglo
        add $s2, $s2, 4 #avanza una posición en el arreglo usando memoria
        add $s0, $s0, 1 #aumenta en 1 el valor del index, que en este caso es a
        j loop
```

Figura 6: Respuesta ejercicio 2.

El código anterior se evaluó con los siguientes valores:

Arreglo inicial	Variable	Registro	Valor inicial	Arreglo final
D: [1,2,3,4,5, 6,7,8,9,10]	a	$s0	0	[4,5,6,7,8,9,10, 11,12,13]
	b	$s1	4	ignorar

Tabla 2: Distintos valores obtenidos en ejercicio 2.

Arreglo inicial	Variable	Registro	Valor inicial	Arreglo final
D: [1,2,3,4,5, 6,7,8,9,10]	a	$s0	0	ignorar
	b	$s1	1	[1,2,3,4,5,6,7,8, 9,10]

Tabla 3: Distintos valores obtenidos en ejercicio 2.

3.3 En el siguiente código se realizó la traducción de C a MIPS de la suma de los pares de un arreglo que se iban almacenando en una variable llamada evensum. A diferencia de los códigos anteriores, en este ejercicio se tuvo que hacer uso de la instrucción “andi” ya que es la encargada de realizar una operación AND entre un registro y una constante inmediata; si esta da 0 al usar como constante inmediata al

1, significa que el número es par ya que analiza el último bit del entero. En la figura se puede ver con más detalle su comportamiento.

```
addi $t0, $zero, 0 #se asume que es evensum
lw $s1, arrlen #lee y guarda el largo del arreglo
la $s0, 0 #se usa la dirección base como index


loop:
        beq $s0, $s1, done #si el index llega al final del arreglo, salta a done
        lw $t2, arr($s0) #lee lo que se encuentra en la dirección de memoria de $s0 y lo guarda en $t2
        andi $t3, $t2, 1 #revisa si el número es par
        beq $t3, $zero, suma #si el resultado de lo anterior da 0 salta a suma, ya que el número sería par
        addi $s0, $s0, 4 #avanza la posición de memoria
        j loop #salta a loop para repetir el ciclo

suma:
        add $t0, $t0, $t2 #suma  el valor del array en cierta posición con evensum
        addi $s0, $s0, 4 #avanza la posición de memoria
        j loop
```

Figura 7: Respuesta ejercicio 3.

Arreglo	Variable	Registro	Valor inicial	Valor final
D: [10, 22, 15, 40]	evensum	$t0	0	72
	i	$s2	0	4
	arrlen	$s1	16	16

Tabla 4: Distintos valores obtenidos en ejercicio 3.

Arreglo	Variable	Registro	Valor inicial	Valor final
D: [7,8,10,12 ,5]	evensum	$t0	0	30
	i	$s2	0	5
	arrlen	$s1	20	20

Tabla 5: Distintos valores obtenidos en ejercicio 3.

Conclusiones

Para finalizar, teniendo en cuenta los objetivos planteados al inicio de este informe, se lograron cumplir la totalidad de ellos, ya que se logra predecir el funcionamiento de los distintos códigos presentados y se hace una correcta traducción del lenguaje C a lenguaje ensamblador MIPS, haciendo un buen uso del simulador MARS y aprendiendo así como usar tal programa para ensamblar y ejecutar códigos de bajo nivel.

Por otro lado, se tuvieron ciertas dificultades al realizar la actividad de traducción, porque se tuvo que aprender a hacer uso correcto de las direcciones de memoria de un arreglo dado y recorrerlo sin pasar el límite de su tamaño, lo que podría provocar errores en todo el proceso a pesar de compilar bien. Este problema logró ser solucionado después de hacer la respectiva traza y junto a ello la revisión de los apuntes entregados por la asignatura.

Al ser el primer laboratorio de la asignatura, se espera usar los conocimientos adquiridos en próximas entregas, ya que lo visto en este informe es la base para lo que se estudiará más adelante en el semestre.

Referencias

Tapia, V. (2024). Lenguaje Ensamblador y Máquina. Organización de Computadores – Primer Semestre 2024.

De Nihil, V. T. L. E. (2019, 9 septiembre). Introducción a la Arquitectura del MIPS. La Cripta del Hacker. https://lacriptadelhacker.wordpress.com/2019/09/09/introduccion-a-la-arquitectura-del-mips/

MARS MIPS simulator - Missouri State University. (s. f.). https://courses.missouristate.edu/kenvollmar/mars/

Sanchez, J. (s. f.). Arreglos y cadena assembler. Scribd. https://es.scribd.com/doc/61311026/Arreglos-y-Cadena-Assembler#:~:text=Un%20arreglo%20es%20una%20lista,declarar%20y%20usar%20los%20arreglos.