DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

ORGANIZACIÓN DE COMPUTADORES

Laboratorio 1: Instrucciones MIPS y Simulación en MARS

Profesor: Viktor Tapia.
Alumno: Rodrigo Pereira Yánez

20 de abril de 2025

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Índice

Introducción.

Este laboratorio explora la programación en MIPS y su simulación utilizando el programa MARS a través de una serie de actividades.

Las actividades de este laboratorio se dividen en tres etapas. La primera busca comprender el funcionamiento del programa MARS para simular código MIPS. La segunda etapa se centra en la predicción manual del comportamiento de programas MIPS, lo que implica comprender y determinar la salida de un código MIPS dado. Finalmente, la tercera etapa consiste en traducir un código escrito en C a instrucciones MIPS, para luego ensamblarlo y ejecutarlo en MARS.

Los objetivos de este trabajo son:

- Predecir el funcionamiento de un programa MIPS.
- Traducir programas escritos en un lenguaje de alto nivel a MIPS.
- Escribir programas MIPS que usan instrucciones aritméticas, de salto y memoria.
- Usar MARS (un IDE para MIPS) para escribir, ensamblar y depurar programas MIPS.

Marco Teórico.

La arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages) nace en la Universidad de Stanford a principios de los años 80 y fue liderada por el profesor John Hennessy.

MIPS fue uno de los primeros procesadores RISC (Reduced Instruction Set Computing), donde la filosofía de RISC es simplificar el conjunto de instrucciones buscando aumentar la eficiencia y velocidad de los procesadores.

La arquitectura MIPS se basa en 4 principios de diseño fundamentales:

1. La simplicidad favorece la regularidad.
2. Hacer el caso común más rápido.
3. Lo pequeño es rápido.
4. Buen diseño demanda buenos compromisos.

El conjunto de instrucciones de MIPS se organiza en diversas categorías, incluyendo:

1. Carga (Load): Transfieren datos desde una ubicación en la memoria principal hacia los registros del procesador para su procesamiento.
2. Almacenamiento (Store): Mueven datos desde los registros del procesador de vuelta a una ubicación específica en la memoria principal.
3. Aritméticas: Ejecutan operaciones matemáticas básicas como la suma, resta, multiplicación y división, almacenando los resultados en registros del procesador.

4. Lógicas: Realizan operaciones lógicas bit a bit, como AND, OR, XOR y NOR, entre operandos y guardan el resultado en un registro.
5. Comparaciones: Comparan los valores de dos registros y, basándose en el resultado, pueden establecer ciertos indicadores o utilizarse para decisiones en instrucciones de salto condicional.
6. Movimiento entre registros: Permiten copiar el valor contenido en un registro a otro.
7. Desplazamiento (Shift): Realizan desplazamientos de bits a la izquierda o a la derecha dentro de un registro, lo que puede utilizarse para multiplicaciones o divisiones rápidas por potencias de dos.
8. Salto y Bifurcaciones (Jump and Branch): Alteran el flujo secuencial de la ejecución del programa. Los saltos incondicionales transfieren el control a una nueva dirección de memoria sin condiciones previas. Los saltos condicionales transfieren el control solo si se cumple una determinada condición (resultado de una comparación entre registros).
9.

La arquitectura MIPS dispone de un conjunto de 32 registros de propósito general, los cuales se clasifican y utilizan convencionalmente de la siguiente manera:

- $0: Valor constante 0
- $at: Registro temporal
- $v0-$v1: Retorno de valores de una función
- $a0-$a3: Argumento de funciones
- $t0-$t7: Cálculo temporal
- $s0-$s7: Guardar variables
- $t8-$t9: Cálculo temporal
- $k0-$k1: Temporales de Sistema Operativo
- $gp: Puntero global
- $sp: Puntero stack
- $fp: Puntero brame
- $ra: Dirección de retorno de una función

En este contexto de la arquitectura MIPS, el simulador MARS (MIPS Assembler and Runtime Simulator) se presenta como una herramienta esencial para el aprendizaje práctico y la experimentación con este lenguaje ensamblador. Es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) específicamente diseñado para facilitar la educación y el aprendizaje del lenguaje ensamblador MIPS. Desarrollado por los doctores Pete Sanderson y Kenneth Vollmar, MARS ofrece un entorno donde los estudiantes pueden experimentar directamente con la programación en MIPS sin la necesidad de hardware físico. Permite visualizar el funcionamiento interno de un procesador MIPS simulado, comprender el flujo de ejecución de los programas de manera detallada (paso a paso) y facilita significativamente la identificación y corrección de errores a través de sus herramientas de depuración.

Desarrollo de la Solución.

Entendimiento MARS

1. ¿Cómo se ensambla un programa MIPS?
    - Primero se debe escribir el código fuente en MIPS, esto se realiza en la ventana de edición de MARS.
    - Teniendo el código escrito, se necesita traducirlo a código máquina. Esto se realiza en el botón que tiene un “Play” con un “martillo” que está etiquetado como “Assemble”. También se puede acceder desde el menú Run>Assemble.
    - Verificar si existen errores en el proceso de ensamblado y de ser así corregirlos. Esto se ve en la ventana “Messages”.
2. ¿Cómo se ejecuta un programa en MIPS?
    - Asegurar de tener el programa ensamblado y sin errores.
    - Hacer click en botón Play que está etiquetado con “Run the current program”.
    - Visualizar los resultados en la ventana RUN I/O.
3. ¿Cómo obtendrías la ejecución en cierta línea que no es necesariamente la última?
    - Identificar la línea que se quiera pausar la ejecución.
    - Establecer un breakpoint.
    - Ejecutar el programa y este se pausará en el breakpoint seleccionado. Aquí se puede examinar el estado actual del procesador, como valores en los registros, contenido de la memoria, entre otras informaciones de interés.
    - Se puede decidir continuar la ejecución paso a paso con el botón “Step” o hacer la ejecución de forma normal con “Run” hasta el próximo breakpoint.
4. ¿Dónde encontrarás el valor actualmente almacenado en el registro $s0?
    - El valor de $s0 se puede ver en la ventana “Registers”, que está en la parte derecha de la interfaz de MARS. Este valor es mostrado en formato hexadecimal, pero es configurable en el IDE para ser visto como decimal.

Resultados.

El apartado referente a la sección de “Predecir el funcionamiento de programa en MIPS.” se encuentra en un documento ANEXO en formato .pdf de nombre “lab1_informe_manual_mips_16610470-k.pdf”

El apartado referente a la sección de “Escribir Programas MIPS” se obtuvieron los siguientes resultados:

El problema era el siguiente:
1. Considera el siguiente código. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado “program1.asm”.

```
C/C++
a = 0;
z = 1;
while (z <> 10) {
       a = a+z;
       z = z+1;
};
```

La traducción del código en C a MIPS quedó la siguiente manera:

```
Unset
.data

.text
       main:
              # Inicialización de variables
              addi $t0, $zero, 0  # a = 0
              addi $t1, $zero, 1  # z = 1

              # while lopp
              while:
                     beq $t1, 10, exit  # (z<>10) -> Mientras z sea distinto
de 10, entra al loop <==> (z = 10) -> Si el valor es 10, va a etiqueta exit
(sale del loop)
                     add $t0, $t0, $t1  # a = a + z
                     addi $t1, $t1, 1   # z = z + 1
                     j while  #Va a etiqueta while
              exit:

              # Fin programa
              li $v0, 10
              syscall
```

Los resultados del código en MIPS son:

- El valor final de $t0 = 45, que en C sería la variable (a).
- El valor final de $t1 = 10, que en C sería la variable (z).
- La implementación de la condición del bucle while (z <> 10) se realizó utilizando la instrucción beq (branch if equal). Esta instrucción efectúa un salto a la etiqueta “exit” si el valor del registro fuente (rs) es igual al valor del registro destino (rt) (ej: beq $rs, $rt, exit). Si bien la lógica del beq se basa en la igualdad, se adaptó para emular la

condición de desigualdad del while. El bucle continúa mientras los valores de z sean distintos de 10, por lo que la lógica implementada salta a la etiqueta de salida (exit) cuando z alcanza el valor de 10, obteniendo así el mismo resultado lógico que la condición original del while.

Conclusiones.

En este laboratorio, se cumplieron los objetivos de aprendizaje sobre programación MIPS y el uso de MARS.

- La predicción manual del código MIPS fortaleció la comprensión de la ejecución y la manipulación de registros.
- La traducción de código de alto nivel a MIPS facilitó la comprensión entre lenguajes y la gestión de recursos a bajo nivel.
- Finalmente, la práctica con MARS facilitó la escritura, ensamblado y depuración de programas MIPS.

En resumen, este laboratorio proporcionó una introducción práctica y fundamental a la programación en MIPS y al entorno MARS, sentando bases importantes para comprender la arquitectura de computadoras.

Referencias.

1. MARS MIPS Simulator. (s.f.). [https://dpetersanderson.github.io/index.html].
2. Patterson, D. A., & Hennessy, J. L. (2014). Computer organization and design: The hardware/software interface. Morgan Kaufmann.