Practica 5 y 6. MIPS IT Simulator - Instalación y Programación

Paul Vazquez
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A00819877

Rodolfo Cuan
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A01233155

1. Introducción

MIPS, ó por sus siglas, Microporcessor without Interlocked Piplines Stages es una familia de microcontroladores con arquitectura RISC desarrollados por la compañía MIPS Technologies [1]. Fueron diseñados en 1984 en el laboratorio de Stanford, por un equipo de investigadores liderado por John L. Hennessy [1]. Quien también fundo la compañía en 1985 [1].

El desarrollo fue bajo la premisa que esta arquitectura es más rápida de hacer ya que es más sencilla que su competencia, los CISC [2]. La principal característica de la familia MIPS son el conjunto de instrucciones y registros que tiene. Los registros son de propósito general y la mayoría de las instrucciones no acceden a memoria, excepto las de carga y descarga [1].

La aceptación que MIPS y su arquitectura son de mayor eficiencia se ha visto reflejando en su popular uso en diferentes aplicaciones como en routers, sistemas embebidos y videoconsolas [3].

2. Desarrollo

Las practicas 5 y 6 fueron trabajadas en conjunto ya que muestran continuidad. Se utilizaron dos nuevos softwares, que fueron ya instalados con la maquina virtual proporcionada. El primero es MipsIT, el cual es una IDE para desarrollar software para la arquitectura MIPS. El segundo es MIPS, el cual es una interfaz gráfica con los elementos necesarios para poder simular nuestro código desarrollado.

2.1. Desarrollo de Parte 1

En esta parte inicial, que corresponde a la practica 5 del manual de practicas se familiarizo con los dos softwares a utilizar [3]. Se creo un nuevo proyecto y se le agrego un archivo prueba proporcionado en la practica. Después de compilar se vio su funcionamiento en el simulador MIPS.

2.2. Desarrollo de Parte 2

Esta segunda parte corresponde a la practica 6 del manual de practicas, donde el objetivo es adentrarnos al código y programar nuestro MIPS [3]. Dentro de esta practica se encuentran una serie de tareas a realizar, las cuales se describirán en las siguiente secciones.

2.2.1. Ensamblador a Código Maquina. La primera tarea a realizar es convertir la instrucción subi $8, $8, 0x2 en código maquina.

Se inicio por encontrar el código de 5 bits la cual representa la operación. Utilizaremos la instrucción *addi* y sumaremos un numero negativo para realizar la resta.

Esta operación es de formato I, la cual tiene la estructura mostrada en la Fig. 1. El opcode correspondiente addi es 001000. RS y RT, son el source y target de la instrucción, como se indica en este caso, ambos son el registro 8. Por ultimo es un numero de 16 bits representando el literal que se le va a añadir, en nuestro caso es -2, el cual va a estar en complemento a dos, 0xfffe. Al final nuestro código maquina completo seria, 0x2108FFFE o 001000 01000 01000 1111 1111 1111 1110.

opcode	rs	rt	IMM
6 bits	5 bits	5 bits	16 bits

Figure 1: Formato tipo I

Después se pide completar un programa que interactué con el switch K2 para controlar el flujo de una operación de suma. Los valores a sumar se leen de los switches ubicados en la ventada Input & Output que corresponde a leer un byte del registro 0xbf90. Si se escribe a dicho registro un byte, se ve reflejado en los LEDS.

Para ello se desarrollaron dos funciones ”wait” y ”wait2’ las cuales esperan que el valor del primer bit de K2 sea ’1’ lógico y ’0’ lógico. Para leer este bit se utiliza una mascara con un AND a 0x01 y luego un branch eq para verificar cuando ya no este en cero.

En wait2 se realiza lo mismo pero con branch not eq. El código que se utilizo es el siguiente:

```
start: jal wait # Wait for button click
nop
jal wait2
nop
lui s0, 0xbf90 #Load switch port address
```

```
lb s1, 0x0(s0) #read first number
nop
jal wait #Wait for button click
nop
jal wait2
nop
lb s2, 0x0(s0) #Read second number
nop
addu s3, s1, s2
sb s3, 0x0(s0)
b start #Repeat all over again
nop
###Add code for wait subroutine here! ###

wait:
        lui s5,0xbfa0
                lb s4, 0(s5)
                nop
                andi s4, s4, 0x00000001
                nop
                beq s4, r0, wait
                nop
                jr ra
                nop

wait2:
        lui s5,0xbfa0
                lb s4, 0(s5)
                nop
                andi s4,s4, 0x00000001
                nop
                bne s4, r0, wait2
                nop
                jr ra
                nop

.end start
```

3. Resultados y Discusión

3.1. Resultado Ensamblador a Código Maquina

Utilizando el código obtenido se ingreso en la memoria ram del simulador y se pudo observar que las instrucciones coincidían con lo esperado (Fig. 8). De igual manera se ve que la memoria maneja 32 bits en forma hexadecimal.

Al ingresar el valor de 0xFF en el localidad de memoria 0x80020000 y corriendo el código se puede ver como la resta se realizo después de ejecutar la instrucción, obteniendo un valor de 0xFD en el registro 8 (Fig. 8). El program counter incremento su valor por 4 al pasar a la siguiente localidad en memoria.

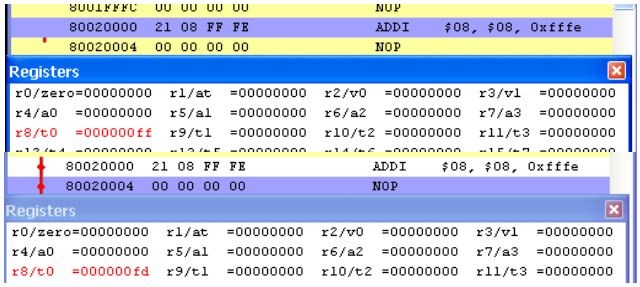

Figure 2: Capturas de Memoria RAM del simulador

3.2. Resultado Practica 6

Para esta practica tuvimos que acceder a las vistas de ”Interrupt I/O” y ”Output” para introducir pulsos por el botón K2, introducir valores con los switches y observar el resultado de la operación en los LEDS. El botón de K2 determina los pasos en el programa, pues al presionarlo se avanza en su ejecución a la siguiente etapa. Para la primera prueba cargamos el valor de ”01”.

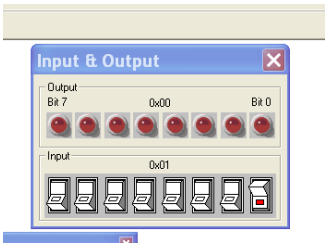

Figure 3: Ingresar 0x01 al programa.

Después ingresamos el valor de 0x02, prendiendo el segundo bit de los switches.

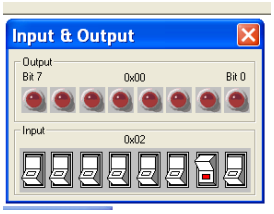

Figure 4: Ingresar 0x02 al programa.

Finalmente obtuvimos el resultado que esperábamos, con un 0x03 en la salida, que es equivalente a prender los dos bits menos significativos de los LEDS.

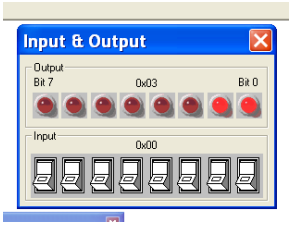

Figure 5: Salida 0x03 al programa.

También decidimos cobrar el caso del de sumar 1 al primer nibble, que es cuando excedemos la cantidad posible de registrar en los primeros 4 bits de los LEDS. Para ello primero introducimos el valor de 0x0F en los switches:

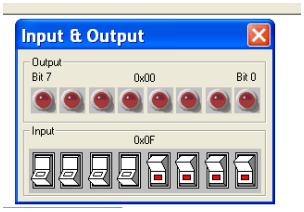

Figure 6: Introducir 0x0F al programa.

Después le sumamos un '0x01':

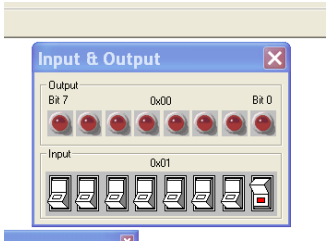

Figure 7: Introducir 0x01 al programa.

Y obtenemos el valor de 0x10 que es el que esperábamos:

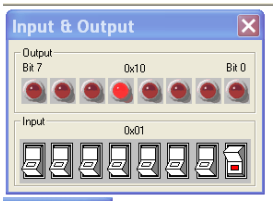

Figure 8: Resultado del programa.

Como ultima prueba decidimos sumar el valor de 0x01 a 0xFF para experimentar un carry out

4. Preguntas del Manual

4.1. Sección 3

a)¿Qué es lo que en realidad se almacena en memoria? Anota las distintas representaciones que toma el contenido de memoria 0x80020000 (ASCII, integer, floating point).

b)¿Cómo se interpreta cada representación?

ASCII: Cada caracter ASCII es de 8 bits por lo que seria 0x80, x0x02, 0x00, 0x00 donde el 0x80 es el simbolo "€" es el espacio " ". Los 0x00 son caracteres nulos.

Integer: Al tener signo equivale a -2,147,352,576.

Floating point: -1.8367

c)¿Cuántos bits observas?

32 bits

d)¿Cómo puede la computadora saber que el patrón de bits almacenado es una instrucción? Al ser el MIPS una arquitectura del tipo Von Neumman, no hay diferencia entre los datos de programa y las instrucciones almacenadas en memoria, el program counter (PC) determina que valor se ejecuta.

4.2. Sección 4

a)¿Cuál es el nuevo valor del 'program counter'? ¿Por qué?

0x80020004, esto fue porque avanzo un registro que son 4 bytes.

b)¿Cuál es el nuevo valor del registro $8? ¿Por qué?

El nuevo valor es 0xFD porque se le resto 2 a 0xFF.

4.3. Sección 5

a)¿Qué hace el programa?

Hace un load a los valores de un switch a un registro y lo lee. Luego llama a una subrutina wait. Después se lee un segundo numero de los switches y se suman. El resultado se escribe en un registro que se conecta a los LEDs.

b)¿Para qué funciona la librería incluida?

Para cargar en el programa las direcciones correspondientes a los registros.

c)¿Qué notas sobre el puerto de I/O?

Que se comunican a través de registros en la memoria.

5. Conclusión

5.1. Paul Vazquez

Con estas practicas combinadas (5 y 6) aprendimos mas sobre el funcionamiento del set de instrucciones en la arquitectura MIPS, y los diferentes tipos de instrucciones que se pueden ejecutar. Algo que exploramos en esta practica que no habíamos tenido la oportunidad de observar el clase, es el uso e periféricos y dispositivos de entrada/salida como los switches y LEDS. Fue importante conocer el so de 'nops' para instrucciones que requieren dos ciclos de reloj en completarse, como branches y jumps.

5.2. Rodolfo Cuan

Con estas dos practicas me queda mucho mas claro como es que funcionan los registros de la memoria de la arquitectura MIPS. Tanto como trabajan con sus puertos, como el direccionamiento de ellos. Fue justo y necesario que estuviera repartida en dos practicas y que nos dieran el código de ejemplo, ya que no me acordaba mucho como programar en ensamblador. Fue retador pero al final se llego al objetivo.

References

[1] Alberto Hernández. *"ARQUITECTURA MIPS"*. Universidad de Valladolid, 2010.

[2] Electronic Design Staff. "Introduction to MIPS Processors". In: (2009). URL: https://www.electronicdesign.com/archive/article/21788878/introduction-to-mips-processors.

[3] Diego Fernando Valencia Martínez. *Manual del laboratorio*. Escuela de Ingeniería y Ciencias Departamento de Ciencias Computacionales, 2020.