Practica 3. Memoria de instrucciones y memoria de datos

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

En la arquitectura MIPS existen dos tipos fundamentales de memorias, la memoria de instrucciones y la memoria de datos. Esta forma de separar las memorias es visto en las arquitecturas Harvard, donde el procesador tiene acceso independiente a cada una de ellas (Fig. 7) [1].

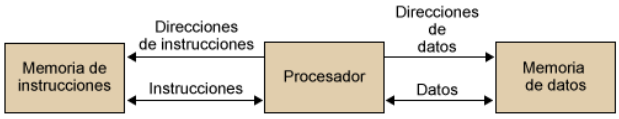

Figure 1: Diagrama del Procesador en Arquitectura Harvard. Imagen extraída de [1]

La primera, la memoria de instrucciones, es de tipo ROM. Como su nombre lo indica, el contenido se conforma por instrucciones en lenguaje maquinal, las cuales son utilizadas por el procesador. La memoria es de tipo ROM, ya que su contenido es constante y no debe ser modificado [2].

La memoria de datos generalmente es de acceso aleatorio y volátil (RAM ó SRAM) [3]. Este tipo de memoria contiene guardado datos en forma de registros, los cuales se utilizan cuando se hace una operación. De igual manera, los datos que se utilizan cuando se esta ejecutando un programa se guardan en esta memoria.

El objetivo de la practica es diseñar y desarrollar los dos tipos de memoria, con las especificaciones presentadas en [2].

2. Desarrollo

La practica fue desarrollada utilizando la maquina virtual proporcionada y el software de ISE Design.

2.1. Memoria de Instrucciones

La memoria de datos descrita en [2], contiene solamente una entrada y una salida: *READ_ADDRESS* e *INSTRUCTION*. Ambas señales son de 32 bits. La memoria misma tiene una capacidad de 32 espacios, los cuales cada uno contiene una palabra de 32 bits.

Para emular los espacios de memoria se opto por el ejemplo presentado en [2]. Esta solución crea un tipo de dato la cual es un arreglo de 32 vectores de 32 bits (Listing 1).

La lectura de la memoria es mediante la señal, *READ_ADDRESS*, la cual indica a que espacio de memoria se quiere acceder. Como se indica en el manual, el direccionamiento de la ROM es por medio de los índices del arreglo, los cuales son números naturales [2]. Por esta razón primero se transformo el valor de hexadecimal a decimal, utilizando la función *to_integer*, de la librería *ieee.numeric_std*. Y después se dividió entre 4 para tener la dirección correcta en la memoria (Listing 2). Las operaciones se realizaron adentro de un process, con la señal de entrada en la lista de sensibilidad.

```
type DATA_ARRAY is array (0 to 31) of
    STD_LOGIC_VECTOR(31 downto 0);

signal ROM_DATA :
    DATA_ARRAY :=(
    OTHERS => (OTHERS => '0')
    );
```

Listing 1: Código VHDL de ROM

```
INSTRUCTION <=
    ROM_DATA(to_integer(unsigned(READ_ADDRESS))/4);
```

Listing 2: Código VHDL de acceso a ROM

2.2. Memoria de Datos

La memoria de datos tiene una estructura interna idéntica a la memoria de instrucciones. La mayor diferencia es que está tiene opción para escribir en ella. La memoria contiene 6 valores de entrada y 1 de salida:

- ENABLE: Entrada que habilita el circuito.
- WRITE_ENABLE: Entrada que habilita el modo escritura.

- READ_ENABLE: Entrada que habilita el modo lectura.
- CLK: Entrada de Reloj
- RW_ADDRESS: Entrada que indica la dirección de memoria para lectura o escritura.
- WRITE_DATA: Entrada que indica los datos para escribir.
- READ_DATA: Salida donde se mostrara la señal leída.

La funcionalidad de las entradas son intuitivas. Para habilitar escritura o lectura, se tiene que activar el correspondiente pin de ENABLE. No se pueden tener los 2 prendidos al mismo tiempo. Para desarrollar esto, se hicieron dos estatutos IFs. Los estatutos buscan cual de estos pines están prendidos, en caso de que uno este prendido se verifica que el otro este apagado (Listing 5).

Cuando el modulo esta en modo lectura se utilizo el mismo método que en la ROM (Listing 2). Si el modo es escritura, el método es similar pero invertido. La dirección en *RW_ADDRESS* se convierte en entero y se divide entre cuatro, para después asignarle el valor de indicado en la señal *WRITE_DATA* (Listing 4). Otras indicaciones adicionales en el modo escritura es la utilización del flaco negativo para escribir y poner la salida en 0s. La primera se concreto poniendo una condición de *falling_edge* para escribir. Toda esta memoria fue puesta dentro de un estatuto IF con la condición que ENABLE este activado.

```
if (READ_ENABLE = '1') and
(WRITE_ENABLE = '0') then ...

//MODO LECTURA

elsif (READ_ENABLE = '0') and
(WRITE_ENABLE = '1') then ...

//MODO ESCRITURA
```

Listing 3: Código VHDL para verificar modo

```
RAM_DATA(to_integer(unsigned(RW_ADDRESS))/4) <=
    WRITE_DATA;
```

Listing 4: Código VHDL de escritura en RAM

3. Resultados y Discusión

Para la elaboración de la practica se pedían realizar las siguientes pruebas:

1) Leer al menos tres localidades distintas de la memoria de instrucciones.
2) Realizar lectura en al menos dos localidades distintas de l a memoria de datos, mostrar el contenido y posteriormente escribir datos nuevos. Al final volver a leer para verificar que el contenido cambió.

Tomando en cuenta que la memoria ROM es solamente de lectura, no es posible realizar el segundo punto en esta memoria.

3.1. Memoria de Instrucciones

Para comprobar que funcionara correctamente, se cargaron previamente 3 valores en la memoria:

```
ROM_DATA(1) <= x"0000ff88";
ROM_DATA(2) <= x"00112233";
ROM_DATA(3) <= x"ff00ff00";
```

Listing 5: Código VHDL para cargar valores en ROM

Que corresponden a la segunda, tercera y cuarta ubicación de memoria, siendo sus direcciones 0x04, 0x08, 0x0C, tomando en cuenta que el program counter va en saltos de 4. Posterior mente se accedió a dichas ubicaciones pasando el valor en READ_ADDRESS.

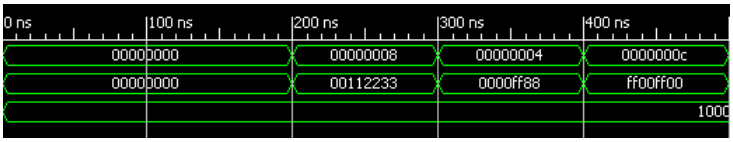

Figure 2: Pruebas ROM

3.2. Memoria De Datos

Para evaluar que funcionara correctamente la memoria de datos primero leemos la cuarta localidad 0x0C, luego la segunda 0x04 y finalmente la primera 0x00:

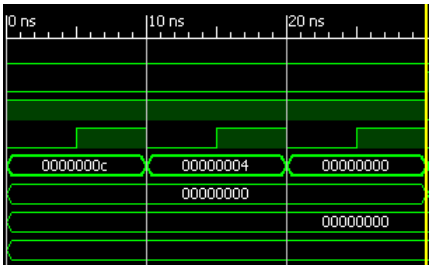

Figure 3: Valor inicial RAM

Y podemos ver que todas las ubicaciones empiezan en 0x00, lo cual es el comportamiento deseado. Posteriormente escribimos y leemos las localidades en ese mismo orden. Primero 0x0C:

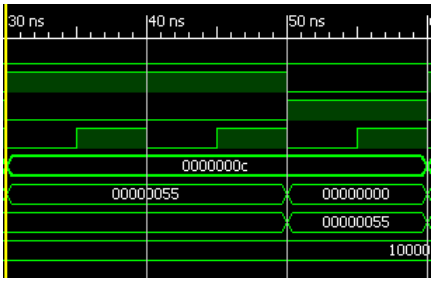

Figure 4: Valor 0x0C RAM

Después 0x04:

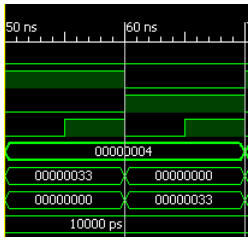

Figure 5: Valor 0x04 RAM

Y finalmente 0x00:

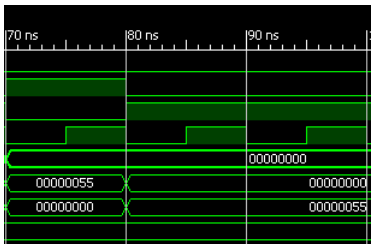

Figure 6: Valor 0x00 RAM

El contenido final de la memoria es el que se esperaba al escribir en esas tres ubicaciones:

	0
0x0	00000055
0x1	00000033
0x2	00000000
0x3	00000055
0x4	00000000
0x5	00000000
0x6	00000000
0x7	00000000
0x8	00000000
0x9	00000000
0xA	00000000
0xB	00000000
0xC	00000000
0xD	00000000
0xE	00000000
0xF	00000000

Figure 7: Valor final RAM

4. Conclusión

4.1. Paul Vazquez

En esta practica considero que desarrollamos una de las partes mas importantes del MIPS, pues ambas memorias tanto la de instrucciones como la de datos, nos permitirán cargar los programas e instrucciones que ejecutaremos, lo cual nos aumenta la capacidad de ejecución del procesador al tener ubicaciones externas a los registros. Se realizo exitosamente y sin complicaciones debido que ambos componentes eran muy similares y el manual nos daba una explicación clara de lo que se requería para que fueran funcionales.

4.2. Rodolfo Cuan

Esta practica no fue complicada, puesto que ambos módulos tenían una estructura similar y con las referencias del manual el desarrollo fue sencillo. Personalmente, me ayudo en la comprensión y diferencia de ambas memorias, y para que se utilizan. Funcionalidades como *to_integer* fueron descubiertas, las cuales son una gran herramienta que puede ser utilizada en las practicas por venir.

References

[1] *El computador Arquitectura Harvard*. http://cv.uoc.edu/annotation/8255a8c320f60c2bfd6c9f2ce11b2e7f/619469/PID$_0$0218274/$PID_0$0218274.$html\#w31aab5c11c21$.

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

[3] *MEMORIA DE DATOS*. http://www.sc.ehu.es/sbweb/webcentro/automatica/web$_8$051/$Contenido/tutor8051_52/Capitulo\%201/memorria_datos.htm$.