Una Simulación del Procesador *MIPS R2000*

José Luis Guzmán Sanchez
Estudiante de Ingeniería Técnica de Informática de Sistemas
Universidad de Almería
e-mail: `jguzman@alboran.ualm.es`

1 Introducción

En esta práctica lo que he tratado de hacer ha sido simular el procesador MIPS R2000 de Silicon Graphics. La informacion para el desarrollo de la misma la he obtenido dellibro *Organización y Diseño de Computadores* de Patterson y Hennessy. Ciertamente la parte que yo he desarrollado ha sido la Unidad de control principal, los multiplexores de 32 bits, la Unidad de control de la ALU, la ALU con anticipación de arrastre y el ensamblamiento de todos los componentes. Además he diseñado una pequeña interfaz con el usuario en modo terminal, en la cual el usuario podrá realizar diferentes operaciones durante la simulación. A continuación ire explicando como he ido desarrollando las diferentes partes y como las he ensamblado.

2 Diseño de la ALU con anticipación de arrastre

En realidad el diseño de la ALU en si es el mismo que teníamos de la biblioteca SLCD++, y lo que yo he mejorado ha sido obtener los bits de acarreo de forma anticipada, de forma que a la hora de obtener el resultado no tenemos que ir calculando de forma secuencial los acarreos como se hacia en la ALU normal. A esta ALU mejorada se le denomina *ALU con anticipación de arrastre*. Para calcular los acarreos he utilizado las siguientes ecuaciones:

$$g_i = a_i b_i$$
$$p_i = a_i + b_i$$

Donde g_i o *generación* será activo si el bit i del sumador genera un CarryOut independiente del CarryIn; y p_i o *propagación* que se activará cuando el bit i del sumador propaga un CarryIn a un CarryOut. Los bits a_i y b_i hacen referencia a los bits de los operandos.

Una vez calculados los bits de generación y propagación para calcular los bits de acarreo habrá que aplicar estas otras ecuaciones:

$$carry0 = g_0 + (p_0 c_0)$$
$$carry1 = g_1 + (p_1 g_0) + (p_1 p_0 c_0)$$
$$carry2 = g_2 + (p_2 g_1) + (p_2 p_1 g_0) + (p_2 p_1 p_0 c_0)$$
$$carry3 = g_3 + (p_3 g_2) + (p_3 p_2 g_1) + (p_3 p_2 p_1 g_0) + (p_3 p_2 p_1 p_0 c_0)$$
$$\ldots$$

Así sucesivamente hasta calcular los 31 acarreos. Una vez calculados bastará con pasarle los acarreos a la ALU normal reduciendo así los ciclos de duración de la misma.
La implementación de la función que calcula los acarreos se encuentra en el archivo *alu32.cc* y se denomina *Carryin*.

3 Diseño de la Unidad de Control de la ALU

La implementación de esta UC es muy sencilla y consiste en 2 puertas AND, 3 NOR y 2 NOT, las cuales y encadenadas de la forma en la que están en la gráfica, tras recibir los 6 últimos bits de la operación (para las R-format) y los bits ALUOPs de la UC principal generan la secuencia de 3 bits correspondiente para indicar a la ALU la función a realizar. La implementación de dicha UC se encuentra en los archivos *uc_alu.** .

4 Diseño de los Multiplexores de 32 bits

Nosotros teníamos multiplexores y multiplexores en la biblioteca SLCD++, pero dichos multiplexores trabajaban con buses de tamaño de 1 bit. Por ello y como nuestros buses son de 32 bits, tuve que ampliar dichos multiplexores. Para dicha ampliación en el caso de los multiplexores de 2 a 1 simplemente tomo los buses de 32 bits y realizo bit a bit la selección, es decir, hago un bucle de 32 iteraciones el cual tendrá en su interior un multiplexor de 2 a 1 de 1 bit, y en cada iteración ire tomando bit a bit de los buses y realizando la selección previamente dicha.
Una vez realizados los multiplexores de 2 a 1, la implementación de los multiplexores de 4 a 1 es inmediata, encadenando 3 multiplexores de 2 a 1 ya adaptados a 32 bits.
La implementación de dichos multiplexores se encuentra en los archivos *muxs.**.

5 Diseño de la Unidad de Control Principal

La UC principal en realidad es una máquina de estados finitos, la cual tras conocer el código de una determinada instrucción va pasando por una serie de estados y generando un conjunto de líneas de control que permitiran poder realizar la ejecución de la instrucción recibida. Para la implementación de esta máquina de estados finitos, podíamos elegir dos caminos diferentes:

1. Implementarla a través de una autómata.

2. Implementarla con una sl PLA(Programmed Logic Array)

Bien, yo me he decido por la segunda. Su implementación es como queda mostrada en la siguiente figura:

De forma que para obtener el resultado de una línea de control hay que ver las líneas verticales a las que hace referencia (señaladas con puntos). Una vez obtenidas, se hará la operación AND de los valores de cada línea vertical; y la obtención del valor final vendrá dado por la operación OR de los valores obtenidos.
Los componentes que forman esta PLA son 54 AND, 10 NOT y 11 OR. Pero para la construcción completa de la UC principal hace falta también un registro de 4 bits que ira guaradando el estado generado. La conexión de la PLA con dicho registro es como sigue a continuación:

Además de los expuesto en el gráfico, para poder considrar cuando llega el final de programa lo que he hecho ha sido introducir una nueva instrucción "end", la cual se intentificará por tener los 6 bits del código operación todos a 1, y una nueva línea de control denominada con el mismo nombre, la cual solo se activará cuando el estado sea el número 10 (un nuevo estado introducido para esta instrucción). La implementación de todo ello se encuentra en los archivos sl uc.* .

6 Conjunto de todos los componentes

Los componenetes utilizados para la implementación del camino de datos son los siguientes:

1. 4 registros de 32 bits.

2. 1 fichero de registros de 32 x 32.

3. 6 puertas AND.

4. 1 puerta OR.

5. 1 UC de la ALU.

6. 1 ALU de 32 bits con anticipación de arrastre.

7. 1 UC principal.

8. El conjunto de todas líneas de control y buses de datos necesarios para la conexión de todos estos componentes.

La declaración de todos estos componentes así como las líneas de control se encuentra en el archivo sl micro.h . Además de todos los componentes que forman el camino de datos en si, también utilizo una memoria de 64x32.

7 Diseño del camino de datos multiciclo

Una vez definidos y declarados todos los componentes la conexión de todos ellos es realizada de forma secuencial. A la hora de la implementación he pretendido llevar el orden aproximado del dibujo de la pag. 287 (del libro que estamos siguiendo) de izquierda a derecha, de forma que los pasos que sigo son los siguientes:

1. Calculo las líneas de control llamando a la UC principal con los bits correspondientes de la operación.
2. Determinación de escritura o lectura del PC.
3. Selección de dirección para el acceso a memoria.
4. Acceso al fichero de instrucción.
5. Acceso al fichero de registros, con la selección previa en los multiplexores.
6. Selección de los operandos de la ALU.
7. Determinación de escritura o lectura en registro Target.
8. Selección de dirección siguiente.

Una vez implementado el camino de datos, lo que hago es comunicarlo con la memoria, de forma que voy leyendo un conjunto de instrucciones previamente cargadas, las cuales trabajarán con un conjunto de datos que también previamente cargados en memoria. Lo que hago en realidad es introducir el camino de datos en un bucle semi-infinito, hasta que se encuentre la instrucción sl end. Una representación gráfica de dicha simulación es la siguiente:

8 Pequeña Interfaz

El simulador ha sido dotado con una pequeña interfaz con el usuario en modo terminal, la cual nos permite poder llevar la simulación paso a paso, ver el contendio de cualquier linea de control o bus de datos, ejecutar el programa hasta el final, interrumpir la simulación etc...Además dicha interfaz nos ira mostrando continuamente la instrucción en curso, y al final de la simulación nos mostrará el contenido integro de la memoria.
Para poder ver los diferentes comando teclear "help".

9 Carja de un programa y ejecución de la simulación

Para poder ejecutar un programa en el simulador bastará con escribirlo en lenguaje ensamblador en un archivo denominado sl prog, y los datos a utilizar en un archivo denominado info.s, donde primero se indicará el numero de datos a introducir y posteriormente se introducirán los datos. Para la ejecución bastará con teclear sl script.

10 Características del procesador

Para el correcto funcionamiento del simulador, me ha sido necesario poner un ciclo de reloj de 22 iteraciones. Bien pues considerando que cada iteración tarda unos 0.9ns, el procesador que estoy simulando tendría una frecuencia de reloj de unos 50 Mhz.