UNA SIMULACION DEL PROCESADOR MIPS R2000

J. A. GIL[1], J. L. GUZMÁN[2], V. G. RUIZ[3]
Departamento de Arquitectura de Computadores y Electrónica
Escuela Politécnica Superior de Ingeniería
Universidad de Almería
Tel: 950 215711
Fax: 950 215486
[1]*e-mail: agilro@hotmail.com*
[2]*e-mail: jlguzman@teleline.es*
[3]*e-mail: vruiz@gogh.ualm.es*

Actualmente, cuando se simula cualquier circuito digital, se intenta conocer su funcionamiento real, ofreciéndonos información sumamente importante para el posterior diseño físico. Llegar a simular un circuito digital nos da ciertas garantías del correcto funcionamiento de su implementación física, aunque dicha garantía viene limitada por la forma de simulación escogida. SDLC++ se presenta como alternativa válida para la simulación del microprocesador MIPS-R2000, esperando de ésta las características propias que los lenguajes de simulación (tiempos de funcionamiento, seguimiento de datos en curso...) y permitiéndonos conocer perfectamente la arquitectura del microprocesador, así como su estructura funcional.

1. Introducción.

En este documento se presenta una simulación del microprocesador MIPS R2000 en su versión multiciclo y reducida, tal y como se describe en el quinto capítulo de [1]. La herramienta software de implementación es la biblioteca de circuitos simulados SDLC++ escrita en C++ y basada en OOP (Oriented Object Programming). Con ella es posible conocer el funcionamiento lógico y temporal de un sistema digital completo.

Gracias a la portabilidad del lenguaje C++ nuestra simulación se ejecuta eficientemente en cualquier computadora que posea un compilador ANSI. En nuestro caso, el entorno de programación ha sido Linux y su compilador g++ de GNU. El software puede descargarse de `http://gogh.ualm.es/vruiz/docencia/sdlc++.tar.gz`.

2. Principios sobre simuladores.

Las representaciones de sistemas digitales en un ordenador son actualmente apuestas claves dentro del campo científico. A través de las simulaciones se pueden obtener datos sumamente importantes que nos ayudarán a facilitar las tomas de decisión sobre el sistema objetivo. Las

más importantes a destacar son: (a) La simulación puede ofrecer datos que nos ayudarán a optimizar el comportamiento del sistema. (b) Nos permite explorar aquellas situaciones en las que la experimentación con el sistema real es peligrosa, problemática, cara o imposible. (c) Nos ofrece una metodología segura y económica para tener un primer contacto con el sistema objetivo. (d) Nos permite adquirir un conocimiento más profundo del sistema simulado. Estas características y otras más son claves importantes a la hora de una implementación física.

3. Estructura del simulador.

La descripción de cualquier sistema digital en SDLC++ se realiza por capas. En la más interna encontramos una serie de objetos que simulan las puertas lógicas básicas (AND, OR, etc...). Con ellos es posible diseñar muchos otros circuitos básicos ((des) codificadores, (des) multiplexores, cerrojos, flip-flops, registros, etc...) de forma jerárquica, hasta llegar a construir con facilidad esquemas relativamente complejos como puede ser el microprocesador MIPS R2000. El modelo usado para simular las puertas lógicas contempla su funcionamiento desde un punto de vista lógico (no eléctrico) y los tiempos de retardo. Por sencillez, todas las puertas tienen asociadas un tiempo de retardo igual a 1 iteración de simulación. De esta forma, es posible conocer si un circuito es más lento que otro simplemente contando el número de iteraciones que han sido necesarias hasta llegar a un determinado estado. Usando SDLC++, la implementación del resto de los circuitos es independiente de la complejidad de los modelos de las puertas lógicas.

4. Aportaciones realizadas a SDLC++

En un principio SDLC++ no contaba con algunos de los elementos funcionales principales que son necesarios para construir el MIPS R2000. En esta sección se exponen nuestras principales aportaciones.

4.1. La unidad de control

La unidad de control es la encargada de gestionar el flujo de datos e instrucciones dentro del microprocesador. En nuestro caso se trata de un controlador de estados finitos compuesto por un registro de estado y una lógica de control combinacional que se puede ver en la página 296 de [1]. Tan sólo hace falta conocer con cierta exactitud cómo está construida la lógica de control combinacional para poder llevarla a cabo. La unidad de control quedaría implementada usando SDLC++ como:

```
void UCM::run(WIRE &clk,WIRE Op[6],WIRE outputs[17]) {
  // Interactuación de datos con el array lógico programable
  pla.run(Op,out_reg,outputs,in_reg);
 // Actualización del registro
  registro.run(clk,in_reg,out_reg);
} //Fin UCM
```

Durante la construcción de la unidad de control se pudo encontrar un fallo en el circuito de la lógica combinacional que es implementado a través de una PLA (página 693 , apéndice C de

[1]). Si la implementación de esta unidad de control no la hubiéramos realizado componente a componente, hubiera sido poco probable haber encontrado dicho fallo. La implementación íntegra de la unidad de control multiciclo se encuentran en los ficheros Ucm.h y Ucm.cc.

4.2. Una ALU con anticipación de arrastre.

Inicialmente se nos proporcionó una Unidad aritmético-lógica con acarreo en serie, es decir, la ALU más sencilla de implementar que existe, pero a su vez la más ineficiente. Esta presenta un serio problema en cuanto al tiempo de ejecución de un programa, ya que al sumar o restar dos valores, según el acarreo generado por la operación a realizar, provoca retrasos de hasta 70 pasos por puerta o iteraciones de simulación en el peor de los casos. Por ello, fue necesaria la implementación de una ALU con anticipación de arrastre donde previamente a la ejecución de la operación se calculan los bits de acarreo correspondientes. Así, se pasa a tener un retraso de 7 iteraciones en el peor de los casos.

4.3. Una Memoria de datos e instrucciones.

SDLC++ inicialmente contaba con una memoria SRAM de 2KB (no caché) de capacidad organizados en 64 palabras de 32 bits de longitud. En nuestra simulación se divide conceptualmente en 2 secciones. La primera se dedica al almacenamiento de instrucciones y comprende desde la dirección 0 hasta la 31. La segunda sección almacena los datos que se utilizan en el programa. Esta memoria es un poco diferente a la usada en [1]. Para resolver este problema de compatibilidad teníamos dos soluciones: (1) construir nosotros una memoria igual a la utilizada en [1] y (2) retocar los caminos de datos del microprocesador y a su vez, si era necesario, también la unidad de control. Finalmente optamos por la segunda. La implementación íntegra de la memoria se encuentra en los ficheros SRAMS.h y SRAMS.cc

4.4. Inclusión de la instrucción end.

Se ha implementado una instrucción especial llamada "end" que indica al simulador cuando debe detenerse. El repertorio de instrucciones que el MIPS R2000 ofrece se puede encontrar en el Capitulo 5 de [1]. La inclusión de una nueva instrucción dentro de este repertorio también se contempla en esta bibliografía, optando como solución añadir una estructura adicional a la unidad de control. Teóricamente es fácil hacerlo, pero su funcionamiento no esta garantizado hasta que se simule correctamente. La instrucción "end" es un claro ejemplo de cómo aumentar la arquitectura del microprocesador y poder explorar por la misma. El resultado de incluir esta instrucción en el repertorio se puede observar en los ficheros UCM.h y UCM.cc

4.5. Interfaz con el usuario.

La interfaz se presenta de forma interactiva con el usuario y es capaz de responder a cualquier consulta sobre el estado de los diferentes buses, líneas de control y contenidos de los dispositivos de memoria (ejm. Memoria de registros) lanzada en cualquier instante de tiempo. También es posible realizar operaciones sobre el flujo de la simulación como la ejecución paso a paso o establecer puntos de parada (breakpoints).

4.6. Un programa ejemplo

A la hora de ejecutar un programa en el simulador, sólo tendremos que escribirlo usando un editor de ficheros ASCII y darle un nombre. Los datos que el programa puede utilizar deben especificarse en decimal y en otro archivo independiente con su respectivo nombre. Para comenzar la simulación se introduce en la línea de comandos: "SR2000 programa.asm datos.asm". Por ejemplo, un programa que multiplica dos números enteros podría ser:

```
//producto.asm
lw $1,32($0)
lw $2,33($0)
lw $4,34($0)
add $3,$0,$0
beq $0,$2,3
add $3,$3,$1
j 4
sw $3,63($0)
end
```

```
//datos.asm
2
3
1
0
```

Donde las instrucciones se almacenan a partir de la posición de memoria cero y los datos se colocarían por orden y a partir de la posición de memoria 32. El resultado sería multiplicar 3*2=6. Nuestra simulación necesita 1539 iteraciones de simulación para alcanzar el resultado final. Tomando el tiempo de retardo asociado a una puerta igual a un 1 nsg el programa se ejecutaría en una máquina real que usara esta tecnología en 1.539 μsg.

5. Conclusión y trabajo futuro

Se ha diseñado un simulador del procesador MIPS R2000 capaz de ejecutar un programa escrito en código máquina. Los resultados que es posible obtener son numerosos y pasan desde validar la arquitectura empleada hasta poder determinar el estado de las líneas físicas en cualquier instante. Gracias a la organización jerárquica de SDLC++, el procesador pasa a ser un objeto más de la biblioteca. Ahora es posible diseñar máquinas multiprocesador y simular así sistemas mucho más complejos con jerarquías de memoria específicas. También hay que realizar un esfuerzo para aumentar el repertorio actual de instrucciones máquina ejecutadas, lo que va a afectar fundamentalmente a la UC. Aunque SDLC++ no incorpora una interfaz de programación gráfica, es cómodo y fácil de usar, especialmente si se desea utilizar los conceptos que la OOP incorpora. Además, ayuda a comprender profundamente los sistemas digitales ya que éstos se describen en última instancia como redes de puertas lógicas. Sin embargo, incluso este último requerimiento puede ser evitado, aunque ahora, los tiempos de retardo asociados al circuito deberán ser indicados de forma explícita.

Referencias.

[1] Patterson, D.A. y Hennessy, J.L. *Organización y Diseño de Computadores. La Interfaz Hardware/Software*. McGraw-Hill. 1995.