PROCESADOR SEGMENTADO

TPE arquitectura de computadoras I

Integrantes:

- Chanes, Juan Manuel(jchanes@alumnos.exa.unicen.edu.ar)
- Lemma, Morena(lemmamore27@gmail.com)

Profesor:

- Goñi, Oscar

Introducción

Se pidió elaborar un procesador MIPS segmentado completo con lo visto en la cursada de la materia, este debe admitir y ejecutar las instrucciones and, or, lw, sw, beq, slt, lui, add, sub. También se pide adelantar el cálculo de la condición y la dirección de salto a la etapa ID junto con un flush cuando el salto es efectivo cuya función es eliminar del pipeline las instrucciones que se encuentran entre la instrucción de salto y el salto efectivo.

Ejecución

- Incorporación de componentes
    - Elaboramos una unidad aritmética lógica(ALU) y un banco de registros con las funcionalidades necesarias para el funcionamiento del MIPS solicitado.
    - Instanciamos las unidades anteriormente nombradas al procesador para poder utilizarlas durante la ejecución de las instrucciones.
- Segmentación
    - Entre cada una de las 5 etapas generamos los registros correspondientes que nos permiten guardar los valores necesarios de las señales.
    - En la etapa IF se obtiene de la memoria de programa provista por la cátedra la instrucción a ejecutar
    - En la etapa ID se decodifica la instrucción y la unidad de control toma los valores de las señales. Además esta etapa calcula la dirección y la condición de las instrucciones beq. También se hace una extensión de signo de los últimos 16 bits de la instrucción.
    - En la etapa EXE se utiliza la ALU para realizar los cálculos correspondientes a cada instrucción(utilizando el campu func en las de tipo R o sumando el valor inmediato con el del registro en el caso de las de acceso a memoria)
    - En la etapa MEM se accede a memoria, ya sea para guardar(SW) o para leer(LW) datos de la misma
    - En la etapa WB se escribe en el registro correspondiente el valor resultante de la ALU o el obtenido de la memoria
- Cálculo de salto
    - En la etapa ID tuvimos que agregar un sumador para calcular la dirección de salto (el valor que teníamos de PCmas 4 y el valor inmediato de la instrucción), también un comparador para hacer un XOR que compare si los valores de los dos registros son iguales, y un AND que actualice el valor de PCin con la nueva dirección en caso de que sea un salto y la condición se efectúe verdadera. Además al valor inmediato de la instrucción tuvimos que multiplicarlo por 4 (shift left 2) para que sea una dirección valida en la memoria de programa.
    - En consecuencia del adelantamiento, la ALU ya no tiene la funcionalidad de calcular la condición de salto, por lo cual la salida Zero no es necesaria.
    - Agregamos un flush para eliminar del pipeline la instrucción que quedó de más cuando el salto fue efectivo.Para la elaboración del mismo, en el proceso reg_IFID, que es el encargado de actualizar los valores del registro,

agregamos una condición que chequea que si no se resetea y hay un salto efectivo, la instrucción del reg se convierte en un nop para limpiar el pipeline.
ID_PcSrc es el selector del mux para saber cual es la próxima dirección del PC, cuando tiene el valor ‘1’ es porque el salto fue efectivo, por lo que hay que borrar la instrucción que fue leída mientras se calculaba el mismo

```
elsif (rising_edge(Clock)) then
    if(ID_PcSrc = '1') then
        IFID_Instr<= x"00000000";
        IFID_PCmas4<=x"00000000";
    else
```

Imagen 1:

Problemáticas

- La gran cantidad de señales nos produjo errores, ya que por desconcentraciones, en un principio, le asignamos diferentes nombres a las mismas señales por lo que no se les asignaba el valor correspondiente.
- El archivo “program1” propuesto por la cátedra generaba riesgos al adelantar las instrucciones de salto, por lo que agregamos NOPs.

Análisis de las mejoras

- El adelantamiento de las instrucciones de salto disminuye el CPI ya que se pierden menos ciclos mientras se calcula y se detecta el salto.

Conclusión

Para concluir con el trabajo podemos reflexionar sobre la importancia de ser organizados durante la elaboración del procesador, ya que el costo de ser desprolijos provoca errores que luego son difíciles de solucionar. Guiándonos con los libros sugeridos por la cátedra, comentando el código, y testeando cada etapa por separado, el desarrollo del procesador se hace más llevadero. En un principio cometimos el error de no testear tan minuciosamente cada etapa y esto provocó que perdiéramos mucho tiempo intentando encontrar los errores que se nos presentaron.