Trabajo Práctico Especial - MIPS segmentado

Breve informe sobre realización, consideraciones y testeos.

Integrantes:

- González Matarras Felipe

Profesor de Laboratorio:

- Dr. Martín Vázquez

Entorno de simulación: Vivado.

Introducción.

El objetivo de este trabajo es implementar el microprocesador MIPS en VHDL. En concreto, se debe realizar la versión segmentada con admisión de las instrucciones add, sub, and, or, lw, sw, slt, beq, addi, andi y ori.

Consideraciones a la hora del desarrollo.

1. Los nombres de las señales están tratados según su etapa, por ejemplo, "EX_MEM_Ctrl_WB" indica la señal saliente de el registro de pipeline "EX/MEM" que es llevada hacia el registro de pipeline "MEM/WB", para posteriormente adquirir el nombre "MEM_WB_Ctrl_WB" que será la señal de habilitación del mux de la etapa Write Back. En cualquier caso, se contiene un diagrama por etapa que contiene una guía con los nombres de las señales asignadas (señales_MIPS.PNG).
2. En la etapa ID, se realiza la instanciación del banco de registros ya realizado anteriormente en la práctica, el archivo tiene el nombre "Registers.vhd".
3. Para diseñar la Unidad de Control, se debía ver la forma de poder indicar las operaciones beq, lui, addi, andi y ori. Para esto fue necesario la incorporación de un bit "extra" en la señal "ID_UCtrl_EX" que luego del pipeline será llamada "ID_EX_Ctrl_EX" para así ir direccionada hacia la "Alu Control". Con este nuevo bit incorporado a la señal, podemos diferenciar adecuadamente las nuevas instrucciones agregadas.
4. En la Alu-Control, en caso de no reconocer ninguna instrucción de una operación conocida, se le manda la señal "EX_ALU_Crl" con '011', representando una acción nula en la ALU.
5. La unidad de control combinacional se diseño con un proceso que es sensible a la señal IF_ID_inst, que según los 6 bits de mas alto peso se determinan los valores de las señales ID_Uctrl_EX, ID_Uctrl_MEM e ID_Uctrl_WB.
6. Los MUX se hicieron en vez de instanciando uno como se hizo con la ALU y el banco de registros, con una asignacion concurrente condicional dado que nos parecio mas simple que tener que mapear cada señal, cuando nos terminaba llevando mas tiempo y espacio que simplemente hacerlo como lo hicimos.
7. Algunas señales como por ejemplo la que debería salir del sign extend en la etapa ID no se modelaron dado que no se usaban en ningun otro proceso o para determinar ninguna otra señal por lo que si bien el registro de pipeline toma el valor que debería no se declara una nueva señal para ello sino que se hizo un proceso. Otro ejemplo de esta decisión es el shift left en la señal de la etapa EX, donde por no ser necesario no se declaro sino que la señal que sale del adder se declara de esta manera:

   EX_MEM_ADD_PC<=(ID_EX_DATAInm(29 downto 0)&"00")+ID_EX_PC_4;
   Donde se puede ver que se podria haber declarado una señal, por ejemplo
   EX_SHL<=(ID_EX_DATAInm(29 downto 0)&"00");
   Y luego se podria declarar el proceso
   EX_MEM_ADD_PC<=EX_SHL+ID_EX_PC_4;

Síntesis y simulación.

Una vez finalizada la etapa de codificación de las etapas, se procedió a correr la síntesis, una herramienta que ayudaría mucho para detectar errores como por ejemplo la falta o sobra de bits en determinadas señales entre otros diversos errores que habían sido filtrados involuntariamente.

Una vez corregidos todos los errores de código, solo quedaba la etapa de simulación, en la que había que corroborar que las instrucciones que se daban en el archivo “program1” se ejecuten de manera correcta en el procesador. Al correr la simuacion de comportamiento se vio que el procesador ejecutaba y que las señales tomaban los valores que debian.

Un problema o percance que se detecto fue que en las operaciones SW si bien la señal de D_DataOut tomaba el valor que debía y que la señal D_WrStb tomaba el valor alto (indicando que deberia escribir) el archivo “data” no se modificaba. Consideramos que es necesario destacar la observación.