Arquitectura de Computadoras 1

Trabajo Práctico Especial

UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS

Integrantes:
Iñaki Martin Arzálluz
Tomas Ezequiel Schuenemann

FACULTAD DE CIENCIAS EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Introducción:

En este informe mostraremos el proceso de desarrollo por el cual se implementó el procesador MIPS segmentado de forma acotada, sin una unidad de detección de riesgos ni una unidad de adelantamientos.
Comenzaremos por la etapa de Fetching con el fin de de llegar hasta la etapa de Write Back, así como también adjuntamos los respectivos diagramas donde haremos hincapié en cómo se fue declarando las señales en los distintos registros de segmentación.

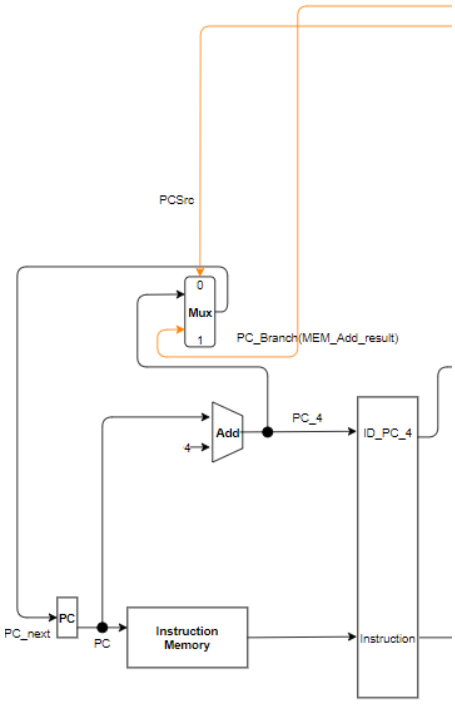
Referencias:

→ Link de entrega:
https://github.com/Beltonidas/TPE_Arqui1_2021

→ Link de imagen de Pocesador:
https://github.com/Beltonidas/TPE_Arqui1_2021/tree/main/_Complemento

Etapa 1: Etapa de Fetching

Se declaró el registro PC, se calculó la potencial siguiente dirección de PC como PC_4 sumando 4 a PC y se eligió el auténtico PC siguiente entre PC_Branch y PC_4 usando un MUX con PCSrc como señal de control.



(Imagen 1)

Etapa 2: Etapa de ID

Definición de la Unidad de control :

Instruction	Regdst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp2	AluOp1	AluOp0
R-format	1	0	0	1	0	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0	0
sw	1	1	0	0	0	1	0	0	0	0
beq	1	0	0	0	0	0	1	0	0	1
lui	0	1	0	1	0	0	0	0	1	1
addi	0	1	0	1	0	0	0	0	0	0
andi	0	1	0	1	0	0	0	1	0	0
ori	0	1	0	1	0	0	0	1	0	1

(Imagen 2) Tabla de valores a asignar en la Unidad de Control.

Cambios respecto de la tabla original:

- AluOp: Se expandió a 3 bits, ya que al tener que diferenciar entre 6 valores distintos, 2 bits no eran suficientes.
- others: ésta no figura en la tabla, pero sería el caso en que venga una instrucción que no pertenezca a la tabla, por lo que las señales de escritura en banco de registros(memtoReg) y en memoria de datos (MemWrite) deben ser “0” para que no se produzcan cambios inesperados en los valores almacenados.

Elección de valores de las señales de control:

- Regdst: Se asignó como “0” para casi todas las instrucciones, ya que estas indica el registro destino en los bit 20 al 16, excepto para las instrucciones de tipo R ya que estas lo indican el registro destino en los bits del 15 al 11. En el caso de sw y beq también se asignó “1” pero simplemente por poner algún valor, ya que de por si en esas instrucciones no se habilitará la escritura al banco de registros.

- ALUSrc: Se asignó como “1” en todas aquellas instrucciones que utilizan el dato inmediato con el signo extendido en la ALU (lw, sw, lui, addi, andi, ori) y como “0” en las demás , ya que tomará el dato que viene del banco de registros (tipo R y beq).

- MemtoReg: Se asignó como “1” únicamente para lw, ya que es la única instrucción que debe tomar el dato obtenido de la memoria de datos.

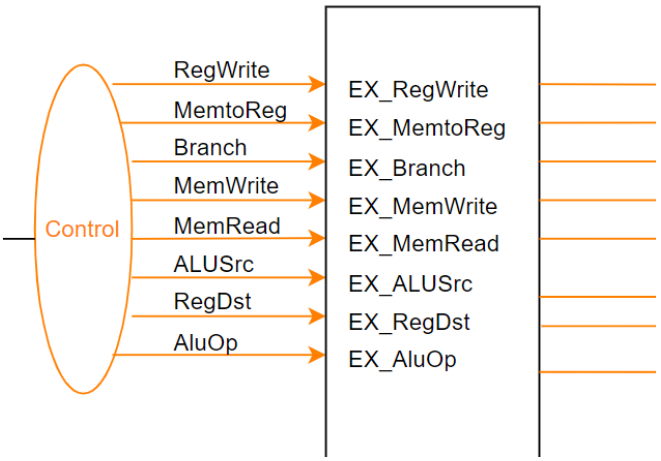
- RegWrite: Se asignó como “1” para casi todas las instrucciones, con la excepción de beq y sq, ya que son las únicas instrucciones que no escriben en un registro.

- MemRead: Se asignó como “1” únicamente para lw, ya que es la única instrucción que debe leer un dato de la memoria de datos.

- MemWrite: MemRead: Se asignó como “1” únicamente para sw, ya que es la única instrucción que debe escribir un dato de la memoria de datos.

- Branch: Se asignó como “1” únicamente para beq, ya que es la única instrucción que puede llevar a realizar un salto.

- AluOp: Se asignó
  - “010” a las operaciones de Tipo-R, y la operación en específico se decidirá en la Unidad ALU Control.
  - “000” a lw, addi y sw, ya que son las operaciones que utilizan una suma.
  - “001” a beq, ya que es la única que utiliza una resta (sub es Tipo-R).
  - “001” a lui, ya que es la única que toma los 16 bits más bajos y los pone en la parte alta.
  - “100” a andi, ya que es la única que utiliza la operación and de la ALU (and como instrucción es Tipo-R)
  - “101”a ori, ya que es la única que utiliza la operación or de la ALU (or como instrucción es Tipo-R)



(imagen 3)

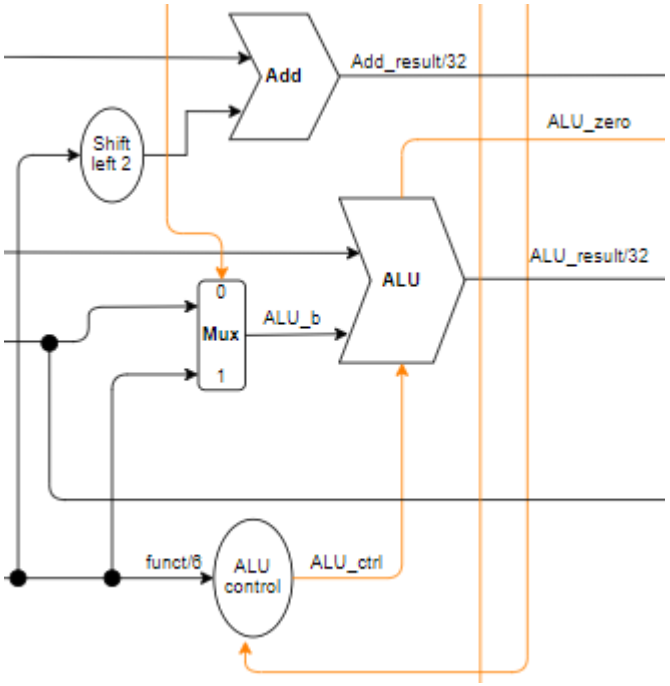
Etapa 3: Etapa de EX

ALUop			funct	ALU_ctrl_2	ALU_ctrl_1	ALU_ctrl_0
0	0	0	x	0	1	0
0	0	1	x	1	1	0
0	1	1	x	1	0	0
1	0	0	x	0	0	0
1	0	1	x	0	0	1
0	1	0	100000	0	1	0
0	1	0	100010	1	1	0
0	1	0	100100	0	0	0
0	1	0	100101	0	0	1
0	1	0	101010	1	1	1

(Imagen 4)

funct: este valor se toma en cuenta únicamente en las instrucciones de Tipo-R, ya que para el resto alcanza con AluOp para reconocer la operación a realizar en la ALU.

Alu_ctrl: En las instrucciones que no son de Tipo-R simplemente se realiza un mapeo desde el valor de AluOp al de ALU_ctrl que se necesite para ejecutar esa operación (esto se habría simplificado si hubiéramos utilizado el mismo valor en ambas señales, pero lo notamos muy tarde). En las instrucciones de Tipo-R se utilizan los 6 bits de menor peso (funct) y se asigna el valor de ALU_ctrl dependiendo de a qué operación corresponde el valor de funct.



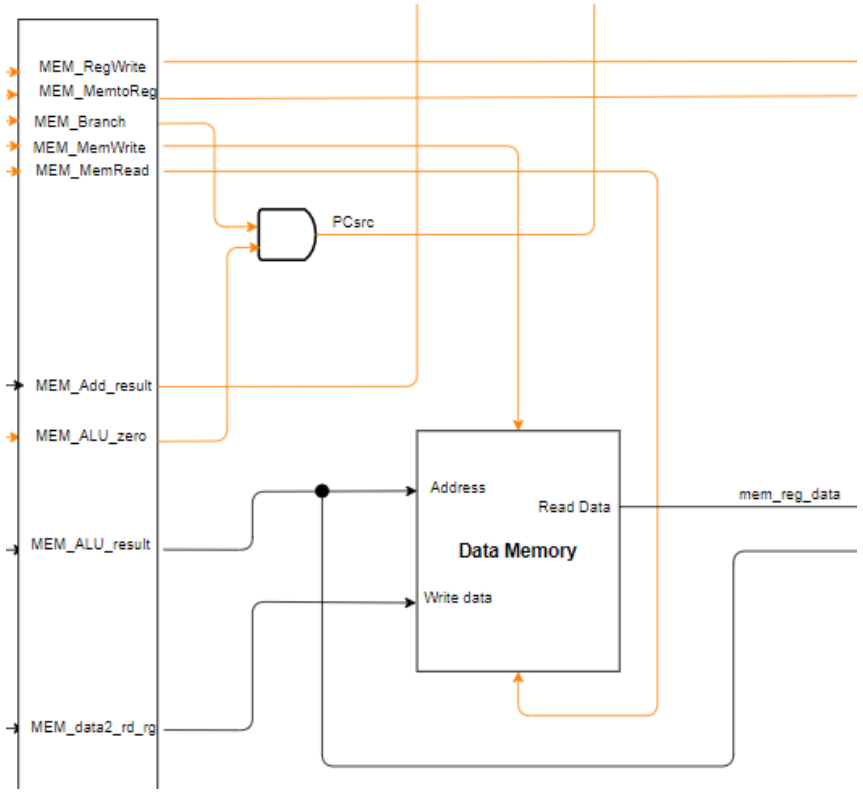
(Imagen 5)

Etapa 4:Etapa de MEM

Elecciones de valores de las señales de Memória:

- MEM_RegWrite: se le asigna el valor anterior de **EX_Reg_Write** que es lo que me habilita el banco de registro para poder escribir en la última etapa en el mismo
- MEM_MemtoReg: se le asigna el valor anterior de **EX_MemtoReg** que es lo que me habilita el MUX de la última etapa para guardar el resultado en el banco de registro
- MEM_Branch: en el caso de un salto posee un '1' y se hace un and con MEM_ALU_zero en el caso de la instrucción beq.
- MEM_MemWrite: activa la escritura en memoria en caso de ser '1'
- MEM_MemRead: activa la lectura en memoria en caso de ser '1'
- MEM_Add_result: resultado de instrucción de salto en el caso de que se produzca
- MEM_ALU_zero: Resultado del ALU si la operación es 0 activa la señal en '1'.
- MEM_ALU_Result:Resultado de ALU en la etapa anterior. Esta es la dirección de la memoria a donde voy a buscar el dato a leer.
- MEM_data2_rd_rg: Se escribe el dato en memoria de un registro seleccionado

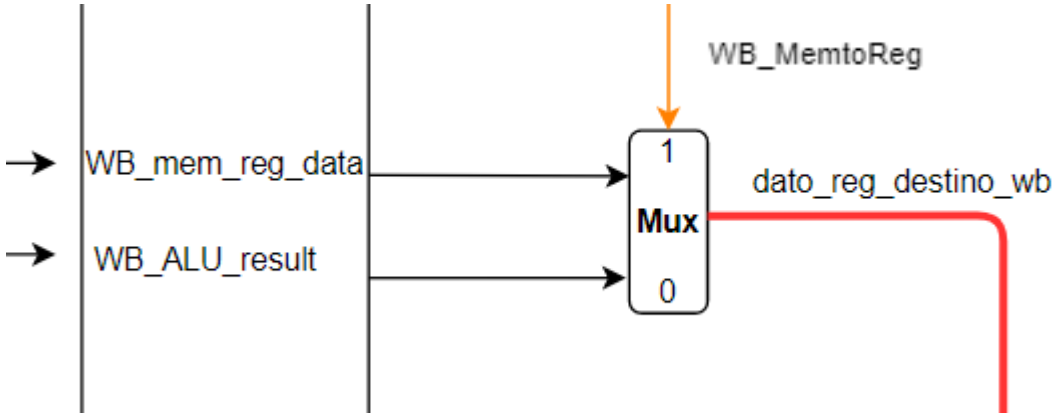
.



(Imagen 6)

Etapa 5: Etapa de WB

Dependiendo de la señal “WB_MentoReg, si es ‘1’ va a escribir en el banco de registros el dato que se obtuvo de memoria y en el caso contrario (‘0’) va escribir en el banco de registros el resultado de la operación que hizo la ALU.(Imagen 7)
También la señal “WB_reg_destino” me va a indicar a qué registro del banco de registro es en el que tengo que escribir



(Imagen 7)

Conclusión:

La implementación del MIPS Segmentado en VHDL nos llevó a una comprensión más profunda de la arquitectura del procesador que no habíamos logrado hasta el momento. A su vez, la mayoría de los errores cometidos se debieron a olvidar las particularidades sobre la concurrencia propia del lenguaje.

Finalmente, esta implementación tiene lugar a mejoras y expansión, ya que no se han implementado las unidades de Forwarding ni de Hazard detection, pero aún así lo consideramos una experiencia positiva, al haber aprendido a utilizar una herramienta muy distinta las utilizadas previamente en otras materias.