UNTREF UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Ingeniería en Computación

Arquitectura de Computadoras 1

Microprocesador MIPS Segmentado

Grupo

Amodey, Leandro – leandroamodey@gmail.com
Araneda, Alejandro – eloscurodeefeso@hotmail.com
Monti, Matias – matiasmonti@hotmail.com

Práctica realizada el:
1er Cuatrimestre de 2019

Docentes:

Prof. Martín Vázquez
Prof. Lucas Leiva

Resumen

La presente práctica de Laboratorio de Arquitectura de Computadoras 1 tiene por objetivo implementar el microprocesador MIPS (*Microprocessor without Interlocked Pipeline Stages*) de arquitectura RISC (*Reduced Instruction Set Computer*) en el lenguaje VHDL (*VHSIC Hardware Description Language*). En concreto, se desarrollará la versión segmentada del microprocesador según el libro de Patterson y Hennessy [1].

Índice General

1. Enunciado

1.1. Consigna original.

Se dispone de la implementación realizada por la cátedra de un procesador completo cuyo *datapath* se esquematiza en la Figura 1 que admite las siguientes instrucciones: `add`, `sub`, `and`, `or`, `lw`, `sw`, `slt` y `beq`. En cualquier caso, la instrucción `beq`, que implica riesgos de control por ser un salto, funcionará "anómalamente" en la versión básica del ejercicio obligatorio.

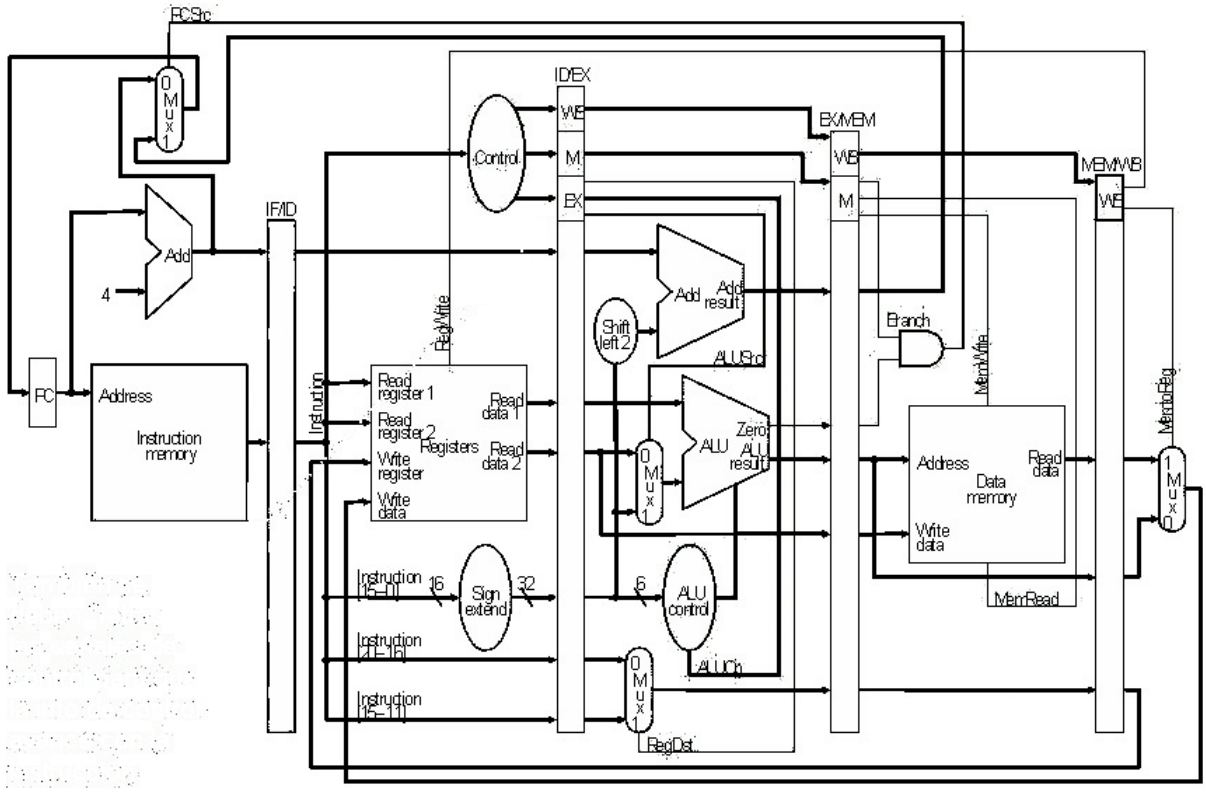

Figura 1. Datapath del procesador MIPS segmentado elaborado por la cátedra.

Se pide que al diseño completo del procesador MIPS segmentado realizado por la cátedra, incorporar las siguientes instrucciones y verificar el diseño utilizando el archivo "`programa`" proporcionado por la cátedra:

Instrucción: `J`
Nombre: *Jump*
Código: **2**
Formato: `J jump_target`
Descripción: `PC ← PC[31:28] & Instruction[25:0] & "00"`

Instrucción: `LUI`
Nombre: *Load Upper Inmediate*
Código: **15**
Formato: `LUI rt, inmediate`
Descripción: `rt ← immediate << 16`

Instrucción: `ADDI`
Nombre: *Addtion Immediate*
Código: **8**

Formato: `ADDI rt, rs, inmediate`
Descripción: `rt ← rs + Sing_extend(inmediate)`

Instrucción: `ANDI`
Nombre: *And Immediate*
Código: **12**
Formato: `ANDI rt, rs, inmediate`
Descripción: `rt ← rs AND Sing_extend(inmediate)`

Instrucción: `ORI`
Nombre: *Or Immediate*
Código: **13**
Formato: `ORI rt, rs, inmediate`
Descripción: `rt ← rs OR Sing_extend(inmediate)`

1.2. Extensión a la consigna original.

Como ejercicio adicional al enunciado de la cátedra hemor implementado la Unidad de Forwarding, o adelantamiento de datos, de manera de abordar también los riesgos de datos propios del procesador MIPS segmentado.

2. Introducción

Los procesadores MIPS son toda una familia de procesadores que siguen la arquitectura RISC: juego de instrucciones reducido, acceso a memoria limitado a instrucciones de carga y almacenamiento, muchos registros de uso general, pocos modos de direccionamiento (inmediato, directo, indexado), y un formato de instrucción homogéneo en longitud y distribución de campos.

Los procesadores MIPS cuentan con una palabra de 32 bits, siendo éste el largo tanto de las instrucciones como el de las direcciones de memoria, y cuentan con 32 registros de generales, los hay del tipo uniciclo (todas las instrucciones se ejecutan en un ciclo de reloj y no se empieza a ejecutar la próxima hasta que no se termina de ejecutar la actual), multiciclo (los distintos tipos de instrucciones pueden ejecutarse en distinta cantidad de ciclos de reloj y tampoco se empieza a ejecutar la próxima hasta que no se termina de ejecutar la actual), y segmentado.

La segmentación es una técnica para optimizar el tiempo de ejecución mediante la separación del procesamiento en etapas determinadas por el uso de recursos independientes de manera de aumentar la cantidad de instrucciones procesadas en el mismo instante pero en distintas etapas. La etapa *Instruction Fetch (IF)* comprenderá el uso del *Program Counter (PC)* y la memoria de instrucciones. La etapa *Instruction Decode (ID)* comprenderá la Unidad de Control y el banco de registros. La etapa *Execute (EX)* incluirá el uso de la Unidad Aritmético Lógica. La etapa *Memory (MEM)* es la correspondiente al acceso a memoria de datos. Y finalmente la etapa *Writeback (WB)* actualizará el banco de registros.

Los procesadores segmentados presentan lo que se denomina **riesgos por dependencia de datos** cuando el operando de una instrucción depende del resultado de otra instrucción precedente que todavía no se ha completado de ejecutar. Una solución a este problema es el adelantamiento de datos, la cual requiere hardware extra como multiplexores, comparadores y buses para adelantar un valor que se encuentra en etapas posteriores del pipeline hacia etapas anteriores.

Los procesadores MIPS poseen tres tipos de instrucciones: las del tipo R, con todos operandos indicados por registros; las del tipo I, con campo inmediato presente en la instrucción; y las de tipo J, de salto absoluto con campo pseudo inmediato presente en la instrucción.

Nos interesa detallar la distribución de campos de las instrucciones de Tipo I, o de campo inmediato:

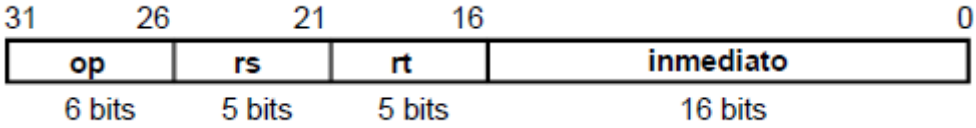

Figura 2. Distribución en bits de los campos de una instrucción Tipo I.

Donde `op` es el campo de código de operación de 6 bits, `rs` es el número del registro origen o fuente de 5 bits, `rt` es el número de registro temporal, e `inmediato` es el campo de 16 bits (o 2 Bytes) que contendrá el valor inmediato o directo que se va utilizar en la ejecución.

Para el caso de instrucciones de tipo J tendremos la siguiente distribución de campo:

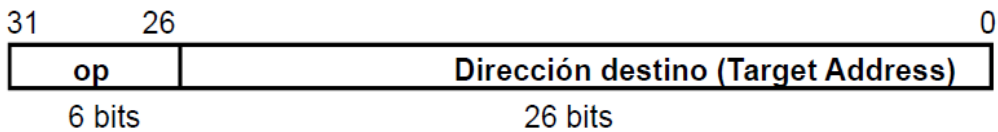

Figura 3. Distribución en bits de los campos de una instrucción Tipo J.

Donde lo primeros 6 bits nos otorgan el código de la operación, y los 26 restantes se utilizan para determinar la dirección de salto junto con la parte alta del `PC`. Asociada a este tipo de instrucción también nos encontraremos con **riesgos de control**, los cuales se generan cuando

tenemos una alteración en la ejecución secuencial de las instrucciones.

3. Recursos utilizados

Para nuestra práctica utilizaremos un entorno de desarrollo completamente de código abierto, basándonos en el software Eclipse para la edición de código y gestión de proyectos, con la extensión ZamiaCAD para la corrección de sintaxis, autocompletado y navegación semántica de entidades, señales y simulación, a la que a su vez integraremos con el compilador GHDL para la elaboración de las simulaciones.

3.1. Entorno de desarrollo.

1.1.1. Instalación del software

A continuación detallaremos los componentes y los pasos para instalar el entorno de desarrollo:

Primero se instalará el software Eclipse [2]. Se debe tener en cuenta que el entorno está destinado para ejecutarse en una máquina virtual del lenguaje Java, por lo que ambas arquitecturas de destino (32 o 64 bits, por ejemplo) deben coincidir.

Como la extensión ZamiaCAD [3] para Eclipse depende del lenguaje Python [4], también lo instalaremos. Se debe agregar la variable `PYTHONPATH` al sistema con el directorio donde se haya instalado el intérprete de Python. También debe agregarse el directorio a la variable `Path` del sistema, al igual que el directorio que contiene los módulos estándares del lenguaje (directorio `Lib`).

Luego será el turno de instalar nuestro compilador externo de VHDL, el software GHDL [5]. De la misma manera que para los componentes anteriores, es necesario agregar a la variable `Path` del sistema el directorio donde se haya instalado el ejecutable (directorio `bin`).

Para la instalación de la extensión ZamiaCAD para Eclipse, debemos iniciar el entorno y seleccionar la opción "`Install New Software…`" del menú principal "`Help`". En el campo del formulario "`Work with`" se ingresará el sitio de actualizaciones de la extensión: `http://zamiacad.sourceforge.net/update-site/`

1.1.2. Configuración del software

Los pasos finales consistirán en la configuración del entorno y nuestro proyecto:

Una vez creado un nuevo proyecto ZamiaCAD en el que importaremos los archivos suministrados por la cátedra, modificaremos el archivo "`BuildPath.txt`" para indicar la entidad descripta en VHDL que es raíz de nuestra jerarquía, en nuestro caso agregaremos la directiva:

```
toplevel WORK.processor_tb(processor_tb_arq)
```

Si el proceso de instalación fue exitoso, se verá reflejado en la consola de ZamiaCAD. En ese caso, sólo falta configurar el compilador externo GHDL, por lo que agregaremos un simple archivo de procesamiento por lotes de comandos del sistema (archivo `build.bat`) como tarea externa completando el formulario que se obtiene al seleccionar las opciones de menú "`Run`", "`External Tools`" y "`External Tools Configurations…`", el apartado "`Program`" y el botón cuya descripción es "`New launch configuration`".

Finalmente, luego de ejecutar la tarea externa y que ésta haya sido exitosa según el resultado de la consola (aparecerán de lo contrario todos los posibles errores en la compilación de los archivos fuentes), configuraremos la simulación completando el formulario que se obtiene mediante las opciones de menú "`Run`" y "`Run Configurations…`", el apartado "`zamiaCAD Simulation`" y el botón cuya descripción es "`New launch configuration`". En el formulario se seleccionará del menú desplegable "`Simulator`" la opción "`VCD Import`", la entidad raíz de la jerarquía "`WORK.processor_tb(processor_tb_arq)`" y la ruta hasta el archivo con la

simulación “processor_tb.vcd”. De haber completado todos los pasos con éxito se tendrá un entorno similar al presentado en la Figura 4.

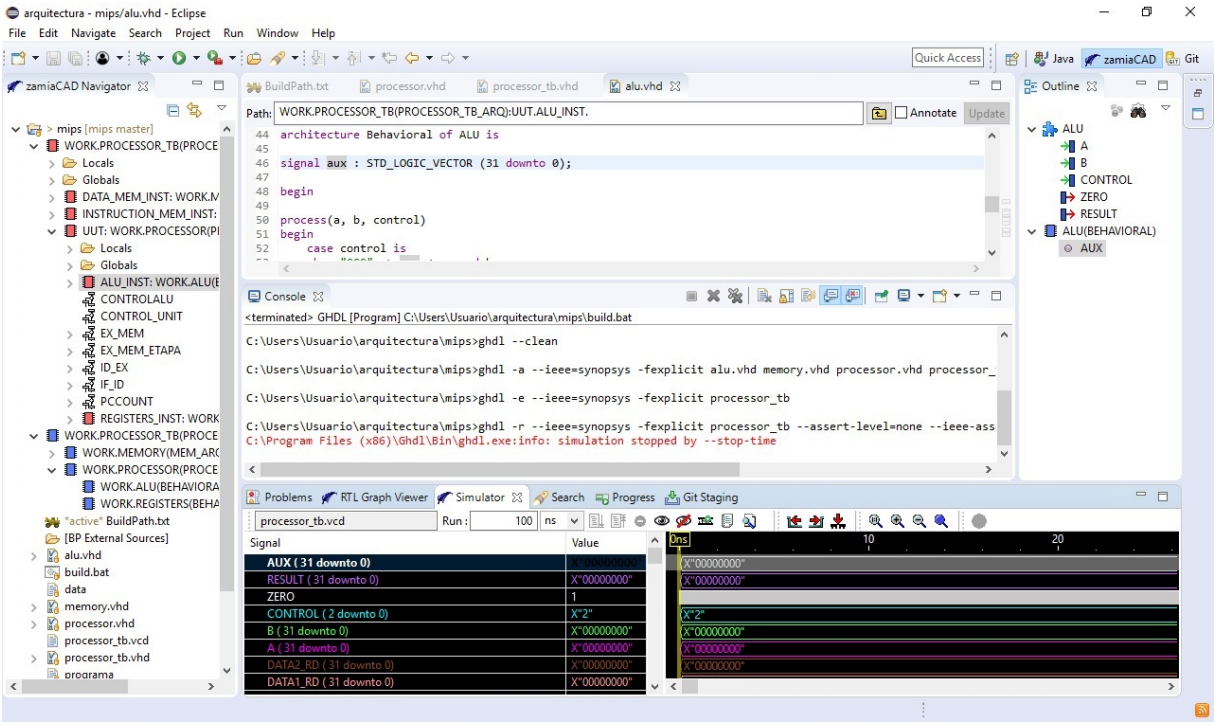

Figura 4. Pantalla de ejemplo del entorno de desarrollo instalado.

3.2. Archivos suministrados por la cátedra.

El programa de prueba “program.s” proporcionado en la práctica no incluye riesgos y prueba todas las instrucciones del ejercicio básico. El archivo “programa” es el resultado del ensamblado del “program.s” que se usará para probar el ejercicio básico. El archivo “datos” contiene los datos que se usaran por el “programa” en el ejercicio básico.

El archivo “memory.vhd” contiene la memoria que se usará en el ejercicio básico. El archivo “processor.vhd” contiene la entidad del micro que se deberá implementar. El archivo “processor_tb.vhd” contiene el test bench para probar el ejercicio básico.

Se entrega además, las descripciones de las entidades alu y registers en los archivos “alu.vhd” y “registers.vhd”, que implementan la unidad aritmetico-lógica del procesador y el banco de registros, respectivamente.

La tabla contenida en el archivo “registers.html” proporcionada en la práctica muestra la traducción de los nombres de registros usados en ensamblador al número de registro en el micro, del 0 al 31.

4. Desarrollo de la solución

A continuación detallaremos las modificaciones al diseño del procesador suministrado por la cátedra para poder incluir las nuevas instrucciones solicitadas por el enunciado de la práctica.

4.1. Extensión del conjunto de instrucciones con ADDI, ANDI, ORI y LUI

1.1.3. Señales y componentes modificados

En la siguiente Figura 5 presentaremos un fragmento del *datapath* en el que se señalan en rojo las señales y componentes que se modificaran para extender el conjunto de instrucciones con la suma aritmética, la conjunción y disyunción lógicas de un registro con un campo inmediato y la carga a un registro de un campo inmediato desplazado a la media palabra más alta.

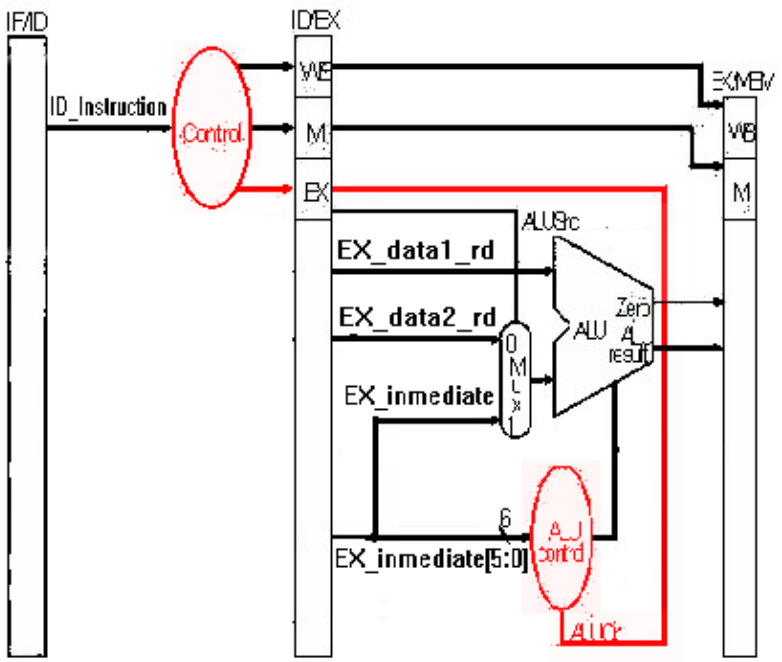

Figura 5. Fragmento del datapath indicando las modificaciones para agregar las instrucciones del tipo I.

Como es posible observar se modificará la Unidad de Control del procesador para que acepte las nuevas instrucciones, obteniendo en consecuencia las señales de control necesarias para las etapas de ejecución, memoria y actualización del banco de registros de cada nueva instrucción.

También se procederá a extender en un bit la señal de control que le indica al Control de la Unidad Aritmético Lógica cual es la operación que se debe realizar. En consecuencia deberá modificarse también dicha entidad para que su lógica combinacional acepte la nueva entrada y le asigne una operación a cada posible valor.

1.1.4. Valores de las señales de control

En la Tabla 1 presentaremos las señales de control que la lógica combinacional de la Unidad de Control del procesador deberá procesar para decodificar efectivamente tanto las nuevas instrucciones agregadas como las modificaciones que es necesario realizar para que continúen funcionando las antiguas instrucciones.

ID_Instruction[31:26]	Nombre	AluOp	AluControl	Operación ALU
000000	R type	**0**10	Decode(EX_immediate[5:0])	calculado
100011	LW	**0**00	010	Suma
101011	SW	**0**00	010	Suma

000100	BEQ	001	110	Resta
001111	LUI	011	100	desplazamiento
001000	ADDI	000	010	Suma
001100	ANDI	100	000	conjunción
001101	ORI	101	001	disyunción

Tabla 1. Valores de las señales de control de las nuevas instrucciones tipo I.

Como es posible observar, reutilizaremos parte de la lógica presente en el procesador diseñado por la cátedra, en particular las señales de control hacia el Control de ALU que permiten las operaciones de suma y resta aritmética, aprovechando las cuatro opciones nuevas que nos provee su extensión de dos a tres bits para agregar tanto las operaciones lógicas de conjunción y disyunción como la operación de desplazamiento a izquierda de dieciséis bits.

4.2. Unidad de Forwarding

Esta unidad es implementada en el archivo "unidad_forwarding.vhd" utilizando la teoría vista durante la cursada. En la Figura 6. Fragmento del datapath con la Unidad de Forwarding.Figura 6 podremos ver un datapath de su funcionamiento sin tener en cuenta que en nuestro diseño además contamos con un multiplexor extra ubicado entre el Mux_B y la entrada B de la Alu que permite usar el dato de inmediate cuando la instrucción actual lo requiera.

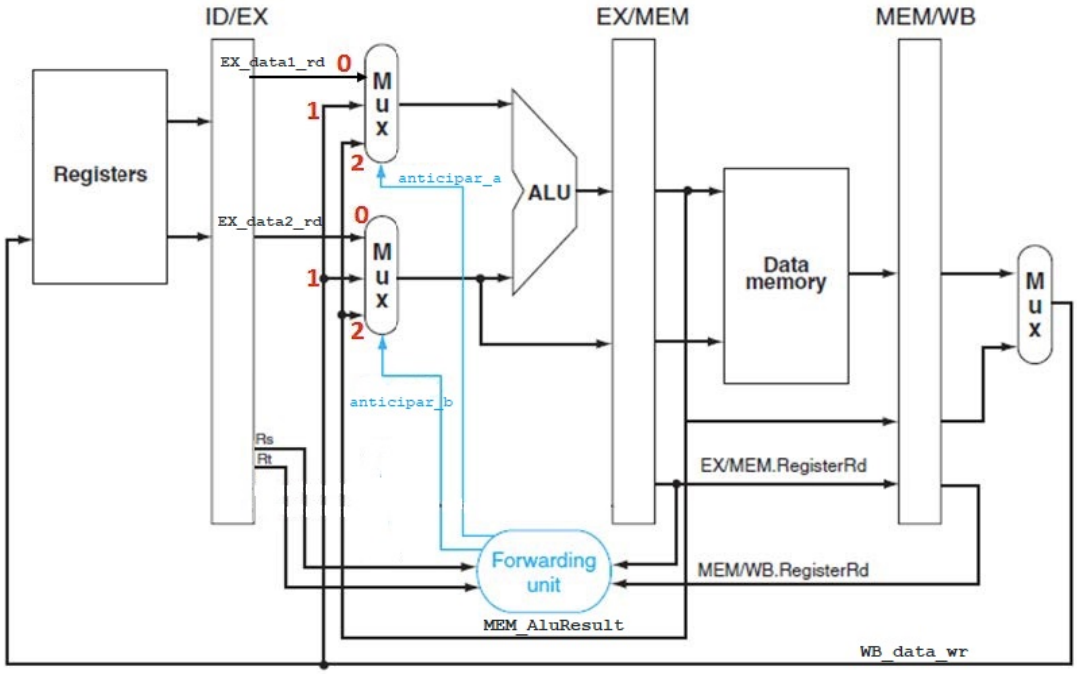

Figura 6. Fragmento del datapath con la Unidad de Forwarding.

El código está basado en el pseudo código descripto en el material brindado por la cátedra, con algunos ligeros cambios en el orden ya que en lugar de separar los posibles errores por etapas, primero se describe la lógica para la salida A de la unidad de forwarding el cual funciona como selector del multiplexor Mux_A, y luego se describe la lógica para la salida B la cual funciona como selector del multiplexor Mux_B.

MUX	Adelanta	Condición	Selector	Señal
Mux_A	Etapa MEM	`ex/mem_rd = id/ex_rs`	"10"	`MEM_AluResult`
	Etapa WB	`mem/wb_rd = id/ex_rs`	"01"	`WB_data_wr`
	Ninguna	Las anteriores son falsas.	"00"	`EX_data1_rd`
Mux_B	Etapa MEM	`ex/mem_rd = id/ex_rt`	"10"	`MEM_AluResult`

	Etapa WB	`mem/wb_rd = id/ex_rt`	“01”	`WB_data_wr`
	Ninguna	Las anteriores son falsas.	“00”	`EX_data2_rd`

Tabla 2. Valores de datos que ingresan a la ALU según la Unidad de Forwarding.

En la Tabla 2 podemos observar los valores que ingresarán a la Unidad Aritmética Lógica una vez que la Unidad de Forwarding haya verificado las condiciones para el adelantamiento de datos, siendo `MEM_AluResult` el resultado de la operación inmediatamente persistido en el registro de segmentación entre las etapas de ejecución y de acceso a memoria, `WB_data_wr` el valor que actualizará el banco de registros en la última etapa del *pipeline*, y `EX_data1_rd` y `EX_data2_rd` los valores leídos del banco de registros en la etapa de decodificación de la instrucción.

4.3. Salto incondicional (tipo J)

Para la implementación del salto incondicional calculamos la dirección de memoria de la próxima instrucción concatenando los cuatro bits más altos del PC, con la dirección del campo `jump_target` desplazado en dos bits a la izquierda. En la Figura 7, se encuentra reflejado el datapath que ejecuta la instrucción dentro del MIPS modificado

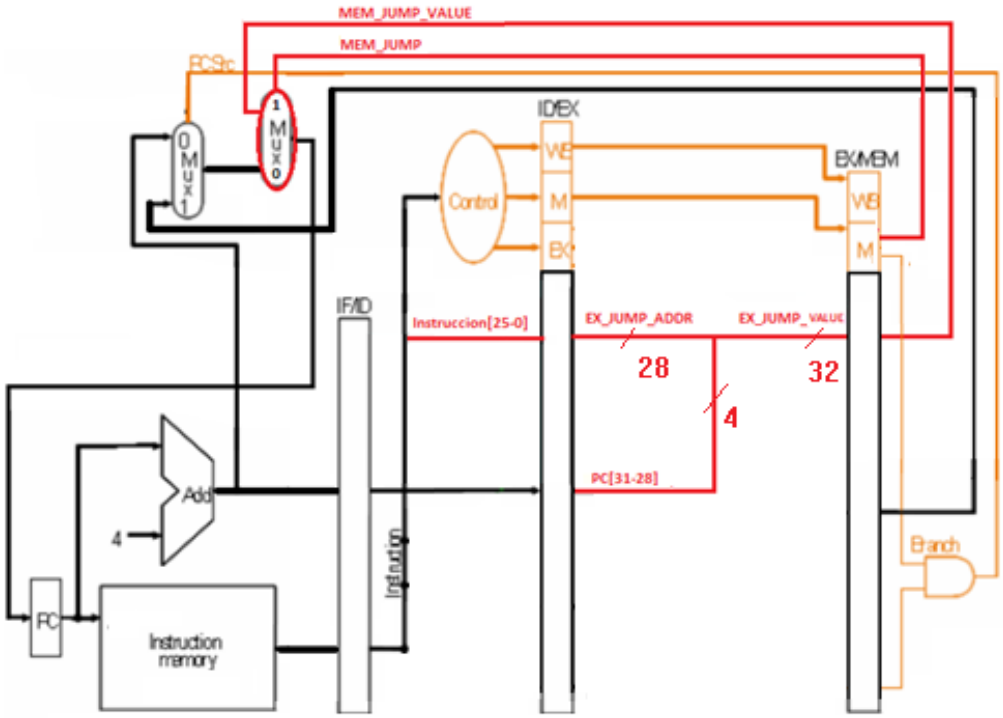

Figura 7. Fragmento del datapath con la lógica adicional para implementar el salto incondicional.

Para la implementación del salto incondicional, añadimos al *datapath* un multiplexor para seleccionar la dirección resultante del `PC` resultante y la señal de control pertinente. Para el cálculo de la dirección del salto, tomamos los 26 bits inferiores del registro J en la etapa ID, lo almacenamos desplazado en el registro ID/EX, y en la etapa Execute lo concatenamos con los 4 bits altos de la señal EX_PC. Finalmente en la etapa MEM, se activa el bit de salto en el multiplexor que indica la siguiente dirección de memoria y se hace el salto efectivo. Presuponemos que el compilador agregará el retardo necesario para tratar con los riesgos de control.

5. Análisis de la implementación

Aquí detallaremos los resultados de nuestra implementación mediante la presentación de simulaciones de partes relevantes del programa proporcionado por la cátedra en forma de gráficas de ondas de las señales pertinentes.

1.2. Extensión del conjunto de instrucciones con `ADDI`, `ANDI`, `ORI` y `LUI`

1.2.1. Elección del fragmento del programa

A continuación presentaremos en la Tabla 3 el fragmento de programa elegido para la simulación que verifique nuestra implementación de las instrucciones con campo inmediato, tanto en su formato de código fuente como en el código compilado adecuadamente agrupado y con la dirección de la memoria de instrucciones correspondiente:

140:	x3C	x0A	x0002	lui $t2, 2
	b001111	b00000	b01010	
144:	x21	x6B	x0006	addi $t3, $t3, 6
	b001000	b01011	b01011	
148:	x35	x8C	x0070	ori $t4, $t4, 112
	b001101	b01100	b01100	
152:	x31	xEF	x0002	andi $t7, $t7, 2
	b001100	b01111	b01111	

Tabla 3. Fragmento de programa elegido para simular instrucciones con campo inmediato.

Entonces podremos comprobar la implementación de nuestras nuevas instrucciones de campo inmediato con las tres últimas del programa. Se proveen los valores hexadecimales de los campos inmediatos para ser comprobados en la simulación (indicados con una “x” adelante), como así también la descomposición en binario del código de la instrucción y los registros operandos (indicada con una “b” adelante).

1.2.2. Simulación del fragmento del programa

Presentamos finalmente la simulación de las instrucciones identificadas en la Tabla 3 del programa tomando las señales ya delimitadas por el fragmento del *datapath* de la Figura 5.

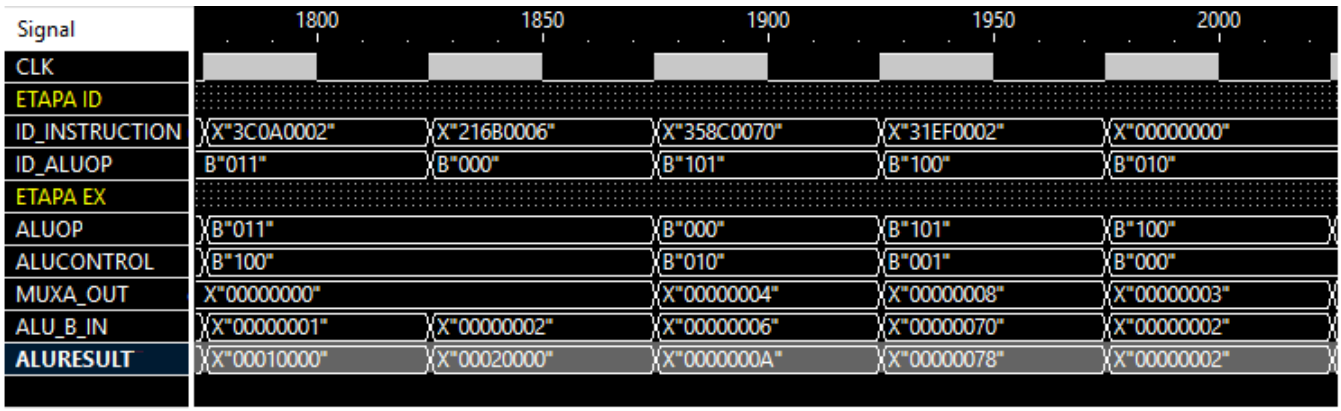

Figura 8. Simulación de instrucciones de campo inmediato

La simulación de la Figura 8 ratifica que la señal hacia el Control de la ALU pasa de la etapa de decodificación a la de ejecución a través del registro de segmentación, que el Control de la ALU procesa esta señal y emite la que finalmente la ALU utilizará para elaborar el resultado, el que es correcto tanto para la suma como para la disyunción y conjunción.

5.1. *Implementación de la Unidad de Forwarding*

1.2.3. Programa compilado para la prueba

Realizado mediante la utilización del *testbench* brindado por la catedra, intercarmbiando el programa de prueba original por "riesgodatos", programa desarrollado y compilado por nosotros, el cual contiene una prueba que nos permitirá comprobar el funcionamiento de la Unidad de Forwarding. En la Tabla 4 detallamos las instrucciones incluidas por nosotros en el test.

0:	x8C	x09	x0000				lw	$t1,	$t0,	0
	b100011	b00000	b01001							
4:	x8C	x0A	x0004				lw	$t2,	$t0,	4
	b100011	b00000	b01010							
8:	x8C	x0B	x0008				lw	$t3,	$t0,	8
	b100011	b00000	B01011							
12:	x8C	x0C	x000C				lw	$t4,	$t0,	12
	b100011	b00000	b01100							
16:	x8C	x0D	x0010				lw	$t5,	$t0,	16
	b100011	b00000	b01101							
20:	x8C	x0E	x0014				lw	$t6,	$t0,	20
	b100011	b00000	b01110							
24:	x01	xEF	x48		x20		add	$t1,	$t2,	$t3
	b000000	b01111	b01111	b01001	b00000	b100000				
28:	x01	xEF	x68		x22		sub	$t5,	$t1,	$t6
	b000000	b01111	b01111	b01101	b00000	b100010				
32:	x01	xEF	x70		x24		and	$t6,	$t5,	$t1
	b000000	b01111	b01111	b01110	b00000	b100100				
36:	x01	xEF	x60		x20		add	$t4,	$t1,	$t3
	b000000	b01111	b01111	b01100	b00000	b100000				

Tabla 4. Programa compilado por nosotros para verificar la Unidad de Forwarding.

Teniendo los valores y las instrucciones a mano podemos observar que no se escribe en ningún registro a la instrucción en la dirección 24, a partir de ahí es donde habrá que analizar si es necesario adelantar algún dato.

1.1.1. Simulación del programa de prueba

En la Figura 9 podremos ver los valores las señales de nuestra unidad de forwarding durante las primeras instrucciones las cuales al ser load dejan los selectores, es de decir las salidas de la unidad en "00" seleccionando el dato que pasaría originalmente si no hubiera unidad de forwarding, esto se debe a que en la lógica la siempre se chequea que el registro RD de las etapas EX/MEM y MEM/WB no coincidan con los registros RS y RT de la instrucción en la etapa ID/EX.

También podemos observar que se realizan dos adelantamientos, en el primero la instrucción en la dirección 28 que tiene habilitada la escritura al ser de tipo SUB, tiene como ID_EX_RS el valor 9 que coincide con el EX_MEM_RD (al adelantar de la etapa EX_MEM el valor del selector será de "10") de la instrucción en la dirección 24 que está en la etapa siguiente por lo tanto la unidad de forwarding adelantara en el MUX_A la señal proveniente de la etapa EX/MEM para evitar utilizar datos erróneos/obsoletos.

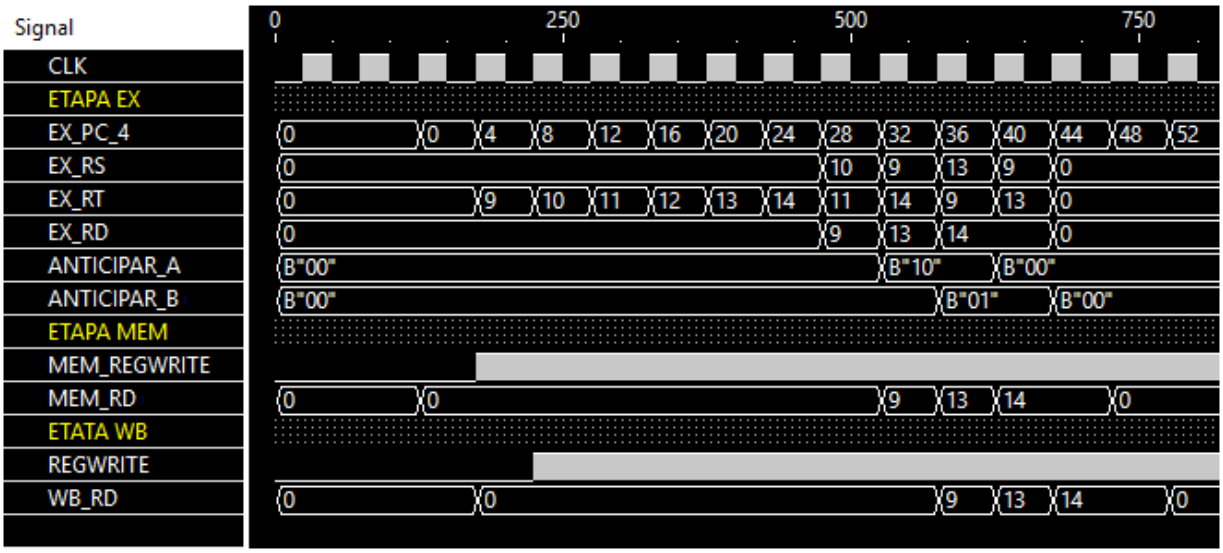

Figura 9. Simulación de la prueba para la Unidad de Forwarding

En el segundo caso, el adelantamiento se producirá en el MUX_B ya que la dirección del registro del conflicto de la instrucción en la dirección 32 es ID_EX_RT, como esta coincide con la dirección del registro RD de la etapa MEM_WB (de la instrucción en la dirección 24) el valor de ANTICIPAR_B será de "01".

5.2. *Implementación de instrucción `j`*

1.2.4. Elección del fragmento del programa

A continuación presentaremos en la Tabla 5 el fragmento de programa elegido para la simulación que verifique nuestra implementación de la instrucción de salto incondicional, con etiquetas y direcciones de la memoria de instrucciones resueltas:

112:	x08 b100011		x000022 b00000000000000000000010110			j inmediatos:	34 (inmediatos)
116:		x00000000				nop	
120:		x00000000				nop	
124:		x00000000				nop	
128:	x01 b000000	xEA b01111	x78 b01010	x20 b01111 b00000 b100000		add	$t7, $t7, $t2
132:	x02 b000000	x0A b10000	x80 b01010	x20 b10000 b00000 b100000		sub	$s0, $s0, $t2
136:	x3C b001111	x00 b00000	x0001 b01001			lui	$t1, 1

Tabla 5. Fragmento de programa elegido para simular instrucciones con campo inmediato.

Podemos comprobar que nuestra implementación asumirá entonces la solución por software del tratamiento de los riesgos de control asociados a la instrucción de salto.

1.2.5. Simulación del fragmento del programa

A continuación presentaremos la simulación del fragmento del programa que incluye la instrucción de salto incondicional implementada por nosotros, incluyendo las señales pertinentes de la Figura 7.

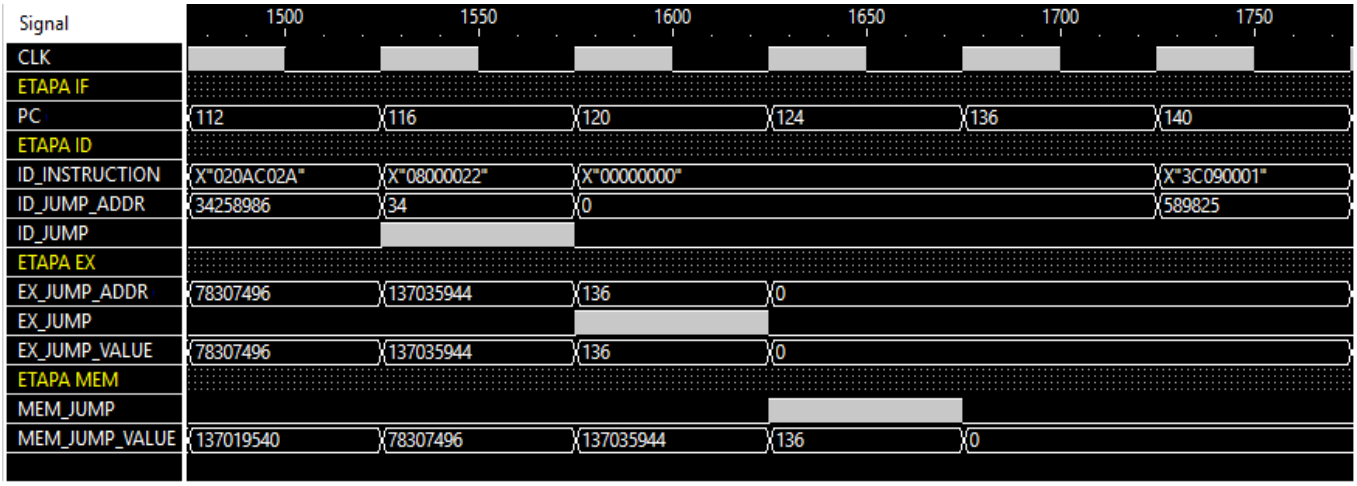

Figura 10. Simulación de instrucciones de campo inmediato

La simulación de la Figura 10 nos muestra como al tercer ciclo después de ingreso de la instrucción de salto en el pipeline del procesador, el PC es actualizado correctamente a la dirección de la memoria de instrucciones correspondiente a la etiqueta del salto.

6. Conclusiones

Consideramos satisfactorio el desarrollo de nuestra solución, conscientes de las posibilidades de ulteriores extensiones, como el adelantamiento de la lógica de salto tanto condicional como incondicional a las primeras etapas del pipeline o la inclusión de una unidad de detección de riesgos que aunque redundante no deposite en el software la solución de riesgos de datos o de control.

La implementación de nuestras extensiones al diseño de la cátedra nos ha obligado y a la vez permitido acceder a un mayor conocimiento de la arquitectura RISC y de los procesadores MIPS. El uso de un lenguaje de alto nivel y las herramientas de simulación hicieron posible una rápida y expresiva comprobación de nuestra práctica.

El entorno de programación resultó apto para administrar la complejidad de la consigna permitiendo no sólo la integración de la codificación con la simulación y el seguimiento estructural del diseño, sino que también facilitó la realización de la asignación de manera grupal. No fue un problema la curva de aprendizaje ya que sus distintos componentes ya fueron introducidos en cursas anteriores como Algoritmos y Programación, Estructuras de Datos y Diseño Lógico.

Como futura referencia, la posibilidad de elaborar mediante software un datapath que dé cuenta de nuestro diseño, sería facilitado por un estilo estructural en la declaración de los componentes del procesador, ya que aún en el caso de herramientas de software privado (Modelsim, por ejemplo) las gráficas resultan poco aptas para la presentación.

Referencias

[1] Patterson, D. A., & Hennessy, J. L. (2009). “Computer organization and design: The hardware/software interface”. Burlington, MA: Morgan Kaufmann Publishers.

[2] “Eclipse IDE. The Leading Open Platform for Professional Developers”. Recuperado de Internet en https://www.eclipse.org/eclipseide/

[3] “zamiaCAD. Open source platform for advanced hardware design”. Recuperado de Internet en http://zamiacad.sourceforge.net/web/

[4] “The official home of the Python Programming Language”. Recuperado de Internet en https://www.python.org/

[5] “GHDL is an open-source simulator for the VHDL language”. Recuperado de Internet http://ghdl.free.fr/