Laboratorio ARMv8 en SystemVerilog

Objetivos

- Desarrollar códigos en lenguaje SystemVerilog para describir circuitos secuenciales y combinacionales vistos en el teórico y el práctico.
- Utilizar la herramienta QUARTUS para analizar y sintetizar el código SystemVerilog.
- Aprender a reutilizar código SystemVerilog mediante módulos estructurales.
- Mediante el uso de test bench, analizar las formas de onda y testear los resultados.

Condiciones

- Realizar el trabajo práctico en grupos de **2 ó 3 personas**.
- Subir el trabajo resuelto a la carpeta de Moodle “EntregaLab1”. La fecha límite de entrega es el **martes 14 de noviembre** (inclusive). No se aceptan trabajos después de esa fecha.

Formato de entrega

- Deben entregar un archivo comprimido (tar, zip, etc.), con el nombre: “Lab1_ApellidoNombre1_ApellidoNombre2_ApellidoNombre3”.
- Utilizar los nombres de los módulos y señales indicados en las guías del práctico.
- Entregar el proyecto de Quartus completo de cada ejercicio resuelto (antes de enviarlo ir a "project/clean project" para eliminar todos los archivos que no son necesarios) y un pequeño informe (**en formato pdf**) que cumpla con los requerimientos descriptos al final de cada ejercicio. Aunque el trabajo es incremental, se pide que se entreguen los 3 ejercicios por separado.
- No está permitido compartir código entre grupos.
- No está permitido subir el código en repositorios públicos.

Calificación

El ejercicio 1 es obligatorio. Su resolución y presentación debe estar aprobada para obtener la regularidad de la materia. Quienes resuelvan el ejercicio 2 (partiendo del procesador modificado en el ejercicio 1) estarán habilitados para promocionar, si cumplen con los demás requisitos. Quienes resuelvan los 3 ejercicios obtendrán 1 punto extra para el primer parcial.

* *Importante*: verificar que no se produzcan advertencias de la herramienta durante el proceso de síntesis relacionadas con una mala interpretación del circuito a implementar.

DESARROLLO

El laboratorio está basado en la implementación de un microprocesador ARMv8 con *Pipeline*, en versión reducida. Para comenzar el diseño, descargar de Moodle el set de archivos <PipelinedProcessorPatterson-Modules> (.sv) con la descripción de los módulos

del procesador con pipeline. A continuación crear un proyecto cuya *Top-level Entity* sea <processor_arm> y agregar los archivos .sv del paquete y todos los archivos de descripción desarrollados en el la guía de práctico n°1. Finalmente verificar las conexiones resultantes según los diagramas de las figuras 1 y 2.

Recomendaciones importantes:

- Recordar inicializar los registros X0 a X30 con los valores 0 a 30 respectivamente en la implementación del bloque *#regfile*.

- Se debe modificar el bloque *#decode* a fin de agregar el puerto de entrada resaltado con un círculo rojo en la Fig. 2.

- Se debe modificar el bloque *#regfile* a fin de que si alguno, o ambos registros leídos por una instrucción en la etapa *#decode,* están siendo escritos como resultado de una instrucción anterior en la etapa *#writeback* se obtenga a la salida de *#regfile* el valor actualizado del registro.

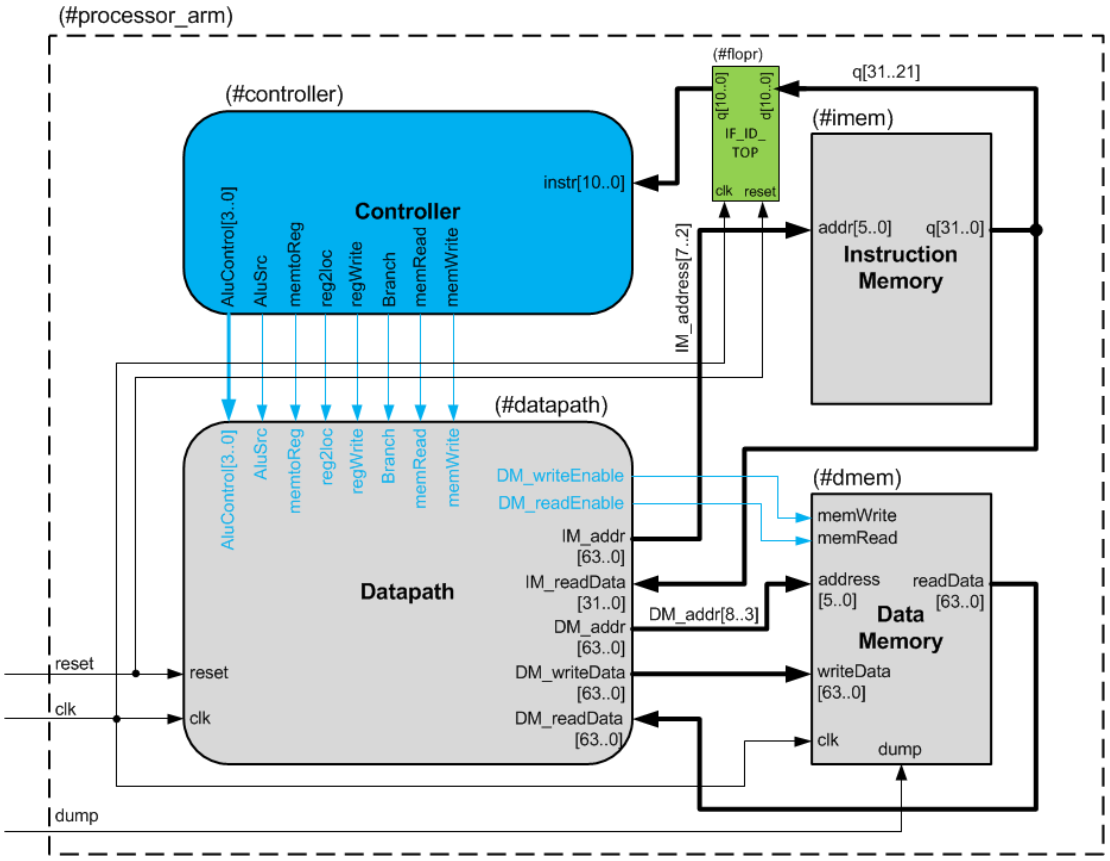

Figura 1: Esquema del *top_level_entity*

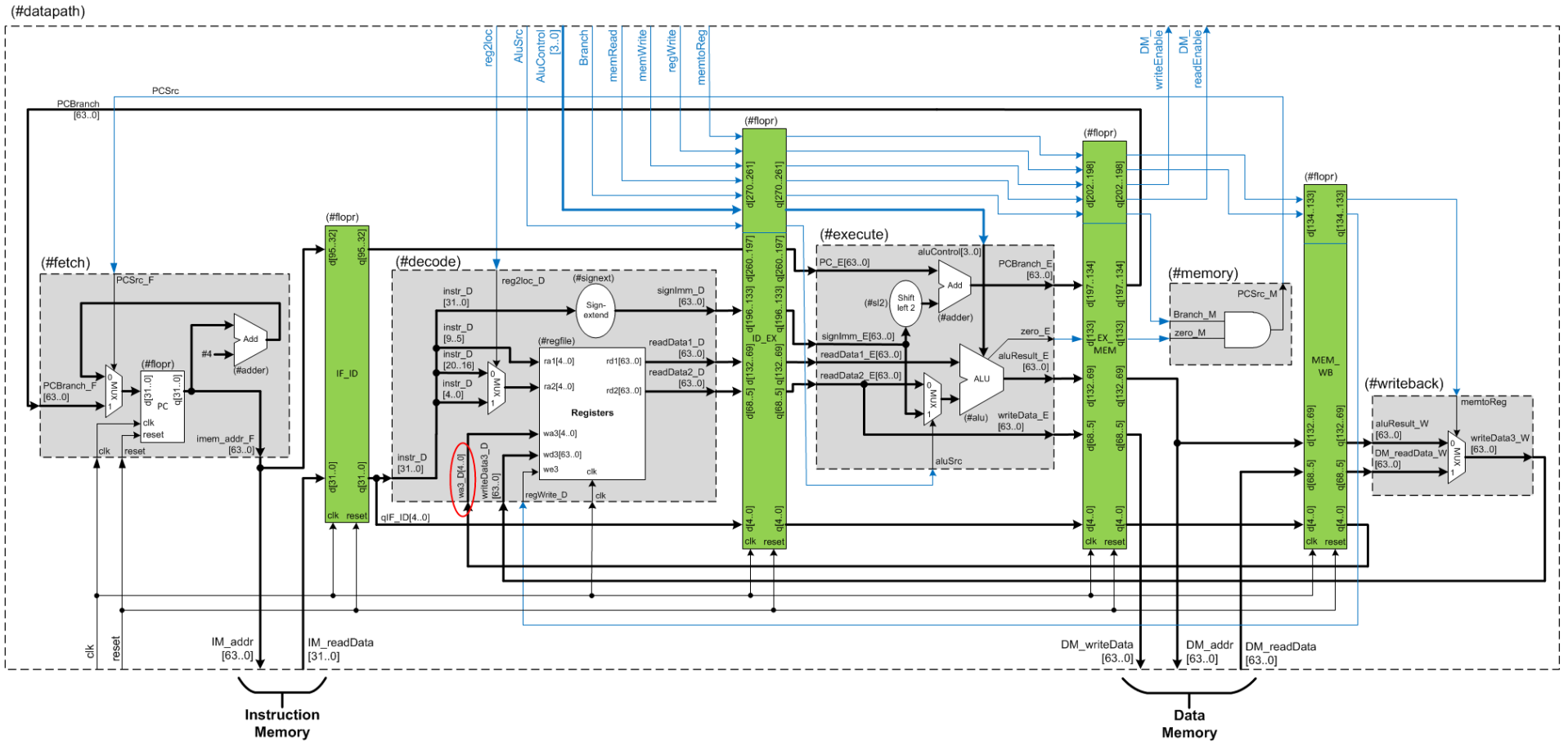

Figura 2: Esquema del *datapath*

Una vez concluida la implementación del procesador completo con pipeline, se debe verificar su correcto funcionamiento utilizando el siguiente código:

```
                                // Dirección:valor
            STUR X1, [X0, #0]     // MEM 0:0x1
            STUR X2, [X0, #8]     // MEM 1:0x2
            STUR X3, [X16, #0]    // MEM 2:0x3
            ADD X3, X4, X5
            STUR X3, [X0, #24]    // MEM 3:0x9
            SUB X3, X4, X5
            STUR X3, [X0, #32]    // MEM 4:0xFFFFFFFFFFFFFFFF
            SUB X4, XZR, X10
            STUR X4, [X0, #40]    // MEM 5:0xFFFFFFFFFFFFFFF6
            ADD X4, X3, X4
            STUR X4, [X0, #48]    // MEM 6:0xFFFFFFFFFFFFFFF5
            SUB X5, X1, X3
            STUR X5, [X0, #56]    // MEM 7:0x2
            AND X5, X10, XZR
            STUR X5, [X0, #64]    // MEM 8:0x0
            AND X5, X10, X3
            STUR X5, [X0, #72]    // MEM 9:0xA
            AND X20, X20, X20
            STUR X20, [X0, #80]   // MEM 10:0x14
            ORR X6, X11, XZR
            STUR X6, [X0, #88]    // MEM 11:0xB
            ORR X6, X11, X3
            STUR X6, [X0, #96]    // MEM 12:0xFFFFFFFFFFFFFFFF
            LDUR X12, [X0, #0]
            ADD X7, X12, XZR
            STUR X7, [X0, #104]   // MEM 13:0x1
            STUR X12, [X0, #112]  // MEM 14:0x1
            ADD XZR, X13, X14
            STUR XZR, [X0, #120]  // MEM 15:0x0
            CBZ X0, L1
            STUR X21, [X0, #128]  // MEM 16:0x0(si falla CBZ =21)
L1:         STUR X21, [X0, #136]  // MEM 17:0x15
            ADD X2, XZR, X1
L2:         SUB X2, X2, X1
            ADD X24, XZR, X1
            STUR X24, [X0, #144]  // MEM 18:0x1 y MEM 19=0x1
            ADD X0, X0, X8
            CBZ X2, L2
            STUR X30, [X0, #144]  // MEM 20:0x1E
            ADD X30, X30, X30
            SUB X21, XZR, X21
            ADD X30, X30, X20
            LDUR X25, [X30, #-8]
            ADD X30, X30, X30
            ADD X30, X30, X16
            STUR X25, [X30, #-8]  // MEM 21:0xA (= MEM 9)
finloop:    CBZ XZR, finloop
```

TENER EN CONSIDERACIÓN LOS EVENTUALES PROBLEMAS DE HAZARD (de datos y de control) QUE CONTIENE EL PROGRAMA UTILIZADO Y PLANTEAR LAS MODIFICACIONES NECESARIAS PARA EVITAR SU OCURRENCIA. Para esto, agregar instrucciones tipo “nop”, las cuales se pueden implementar como ADD XZR, XZR, XZR.

Ejercicio 1 (obligatorio)

Sin afectar el funcionamiento de las instrucciones ya implementadas en la versión reducida del microprocesador con pipeline, **agregar** la instrucción **MOVZ**. Introducir en el procesador todas las modificaciones necesarias, tanto en el datapath como en las señales de control.

Las especificaciones para la implementación de esta instrucción pueden obtenerse de la Reference Data LEGv8 (GreenCard LEGv8) del libro “Computer Organization and Design - ARM Edition”, y se resumen a continuación:

- MOVZ: “Move wide with zero”. Instrucción de carga de valores inmediatos en un registro. 16 bits del registro destino (Rd) toman el valor dado por MOV_immediate y el resto de los bits toma valor ‘0’. La posición del valor inmediato se determina con los 2 bits de LSL, según el siguiente criterio:

LSL	Bits que toman el valor inmediato
00	0 a 15
01	16 a 31
10	32 a 47
11	48 a 64

Sintaxis: `MOVZ` ***<Rd>, <MOV_immediate>, <LSL>*** Tipo: IW

31 23	22 21	20 5	4 0
opcode	**LSL**	**MOV_immediate**	**Rd**
1 1 0 1 0 0 1 0 1	L L	X X X X X X X X X X X X X X X X	R R R R R

Una vez finalizado, agregar al código dado (en la página anterior) instrucciones MOVZ con distintos valores numéricos y de LSL y verificar su correcta implementación. Es decir, se debe analizar que todo el set de instrucciones continúe funcionando correctamente y también los MOVZ (para esto pueden escribir los resultados en memoria y verificar si obtienen los valores correctos en “mem.dump” al finalizar la ejecución del código).

Para el informe:

- Describir brevemente qué modificaciones se introdujeron (en qué entidades y con qué finalidad). Mostrar el diagrama del nuevo microprocesador, indicando las señales y entidades agregadas (de ser necesario).
- Mostrar el programa en assembler LEGv8 modificado que se utilizó para verificar el comportamiento del procesador.

Ejercicio 2 (para promocionar)

El procesador LEGv8 desarrollado no tiene la capacidad de detectar la ocurrencia de *hazards* de ningún tipo. En este ejercicio se propone la implementación de un bloque de detección de *hazards* (*Hazard Detection Unit*) y otro de *forwarding (Forwarding Unit),* a fin de aplicar la técnica de *forwarding-stall* en caso de la ocurrencia de un ***data hazard****,* hasta que el mismo desaparezca.

Algunas aclaraciones respecto a la implementación:

- La *HDU* debe implementarse en la instancia del *Instruction decode* (ID).
- Las diversas condiciones para la detección de un *data hazard* se analizan en el capítulo 4.7- “Data Hazards: Forwarding vs Stalling” del libro “Computer Organization and Design - ARM Edition” de D.Patterson y J. Hennessy. Se deben considerar TODAS las condiciones para todos los tipos de dependencias de datos.
- Para generar la condición de *stall* en el procesador es necesario:
    - Evitar que el PC avance a la siguiente instrucción en el siguiente CLK y evitar que el registro de pipeline IF/ID cambie de valor en el siguiente CLK (congelar su valor). Para esto deberán diseñar una nueva entidad **FLOPRE** similar al **FLOPR**, pero agregando una señal de *enable* (habilitación). Funcionamiento: *enable* = 1 el funcionamiento es normal, *enable* = 0 no altera el valor de salida al detectar un flanco de CLK (síncrono).
    - Forzar que todas las señales de control a partir del ciclo EX en adelante tomen el valor “0” (Ver implementación de referencia en Fig 4.59).
- ¡No olvidar que una parte del registro IF/ID está en la entidad *#processor_arm*!

Para el informe:

- Mostrar un diagrama de bloques general del nuevo procesador, indicando las señales y módulos agregados.
- Correr nuevamente el código dado en el ejercicio 1, sin agregar instrucciones nop, y tomar una captura de la pantalla “Wave” de ModelSim donde se vea un caso de *stall* de instrucciones y otra donde se vea el *forwarding* de datos entre instrucciones. Explicar brevemente qué se observa en la imagen.

Ejercicio 3 (+1 punto)

Se desea incorporar al procesador con *forwarding-stall* la lógica necesaria para detectar la ocurrencia de un ***control hazard*** y hacer *flush* de las instrucciones cargadas incorrectamente en el pipeline.

Algunas aclaraciones respecto a la implementación:

- La señal *PCSrc_M* sólo toma valor “1” en el caso que un salto deba ser ejecutado, por lo que se puede utilizar esta señal para identificar la ocurrencia de un hazard de control.
- Al encontrar una condición de hazard de control verdadera, se debe generar el *flush* de las instrucciones cargadas erróneamente en el procesador, es decir, limpiar las señales de control de los registros IF/ID, ID/EX y EX/MEM. Para esto deberán:

    - Transformar la salida de la *Instruction Memory* (en la entidad *#processor_arm*) en una instrucción NOP, colocando ceros en los 32 bits.
    - Forzar que todas las señales de control de los registros ID/EX y EX/MEM tomen el valor “0”.
    - En todos los casos deben interponer multiplexores entre las señales actuales y los ceros a introducir, controlados por la señal *PCSrc_M*.
- ¡No olvidar que una parte del registro IF/ID está en la entidad *#processor_arm*!

Para el informe:

- Mostrar un diagrama de bloques general del nuevo procesador, indicando las señales y módulos agregados.
- Correr nuevamente el código dado en el ejercicio 1, sin agregar instrucciones nop, y tomar una captura de la pantalla “Wave” de ModelSim donde se vea el *flush* de instrucciones. Explicar brevemente qué se observa en la imagen.