Práctico N° 2: Test del procesador de un ciclo

Antes de comenzar, es necesario clonar el repositorio (seguir las instrucciones dadas por Zulip). En el aula virtual van a encontrar el archivo comprimido "armv8 createAssembly" que contiene el template de un archivo ASM y su makefile para generar binarios a partir de código assembler ARMv8. También van a encontrar un archivo comprimido llamado "olla" (Opcode Loader for Legv8 Architecture) que contiene un script que facilita la inicialización de la imem con el código implementado.
En el repositorio ya está creado el proyecto llamado processor_arm, en la carpeta módulos está la descripción de los módulos del procesador, conforme a los esquemas de las Fig. 1 y Fig. 2 que completan la implementación del microprocesador en conjunto con los desarrollados en la guia 1. Agregar en esta carpeta dichos bloques desarrollados por ustedes y verificar las conexiones resultantes según los siguientes diagramas:

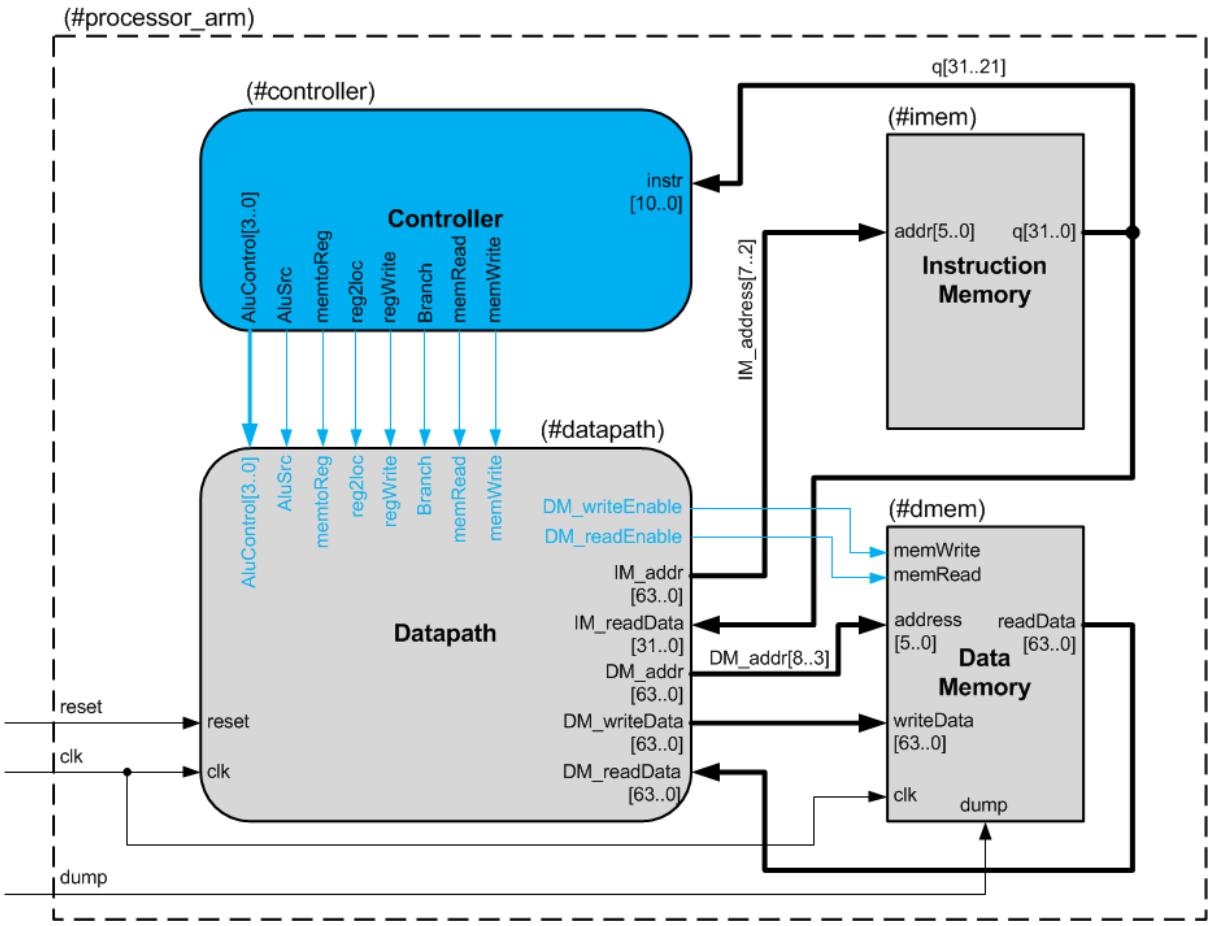

Figura 1: Top Level ARM processor

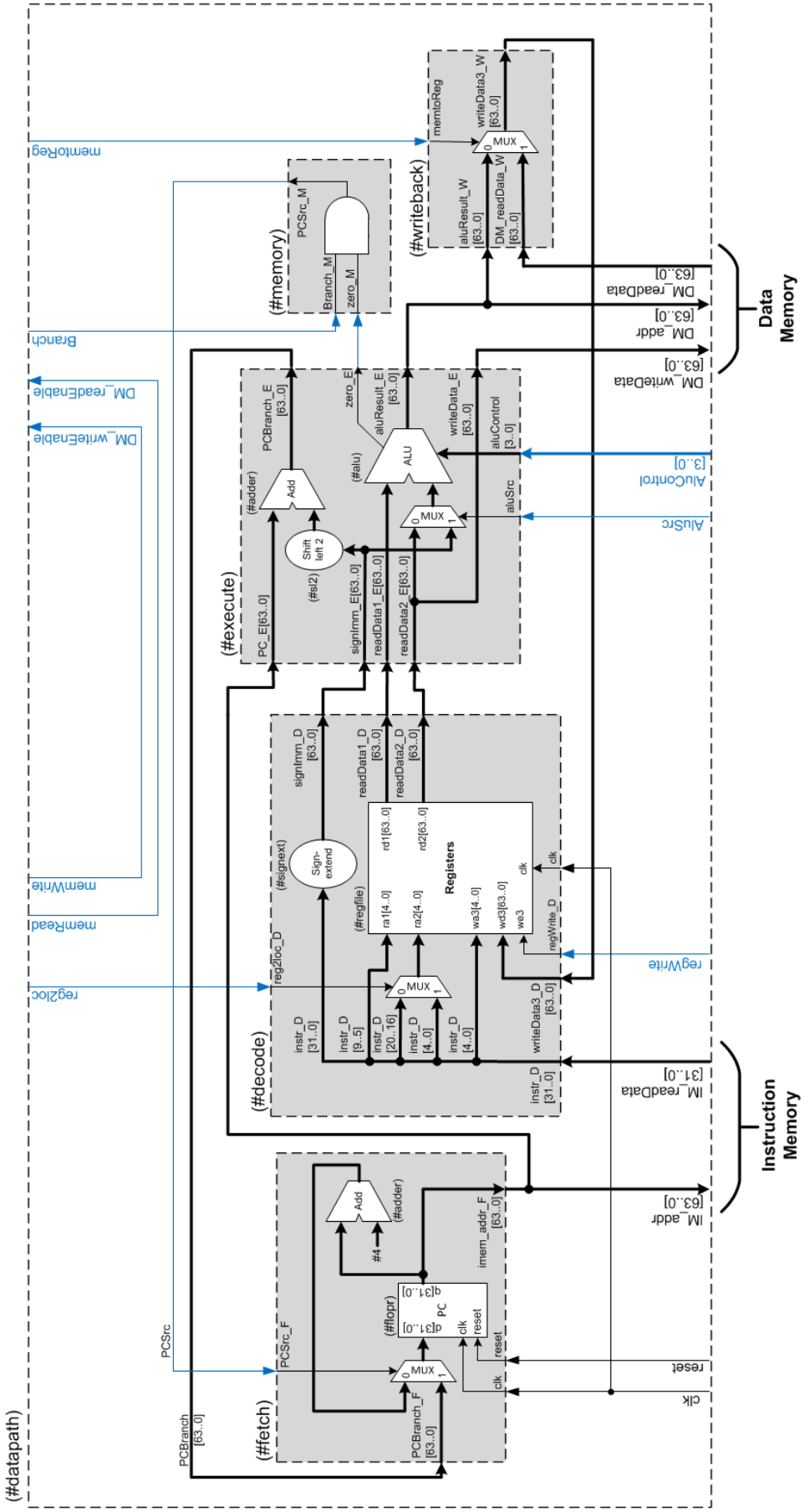

Figura 2: Datapath ARM processor

Ejercicio 1:

Asociar el test bench suministrado <processor_tb>, compilar el proyecto (solo análisis y síntesis) y correr la simulación RTL. La memoria de programa (imem) diseñada en la guía 1 contiene precargado un programa que prueba (de manera no exhaustiva) todas las instrucciones del microprocesador. Para asegurar el correcto funcionamiento del procesador analizar que el archivo “mem.dump” que se genera en la carpeta *...\simulation\modelsim* tenga el contenido esperado, según se indica en los comentarios del programa:

```
                                    // 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, loop1
            STUR X21, [X0, #128]    // MEM 16:0x0(si falla CBZ =21)
loop1:      STUR X21, [X0, #136]    // MEM 17:0x15
            ADD X2, XZR, X1
loop2:      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, loop2
            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
```

Nota: El archivo “mem.dump” contiene las direcciones de la memoria de datos (columna izquierda) y su contenido (columna derecha). La memoria se inicializa con ceros y los datos guardados corresponden a la ejecución del programa de prueba.

Ejercicio 2:
Escribir programas en assembler LEGv8 en base a los siguientes enunciados, con el fin de corroborar el correcto funcionamiento del procesador implementado:

2-A) Con la menor cantidad de registros e instrucciones, inicializar con el valor de su índice las primeras N posiciones de memoria (comenzando en la dirección “0”).

2-B) Realizar la sumatoria de las primeras N posiciones de memoria y guardar el resultado en la posición N+1.

2-C) Realizar la multiplicación de dos registros: X16 y X17 y guardar el resultado en la posición “0” de la memoria.

A TENER EN CUENTA:

- Para este ejercicio se podrán utilizar SOLO las instrucciones LEGv8 soportadas en el procesador implementado en HDL (LDUR, STUR, CBZ, ADD, SUB, AND, ORR).
- Para obtener el código ensamblado en hexadecimal, se debe escribir el programa a implementar en el archivo “main.s” del paquete <armv8_createAssembly>, luego escribir por terminal:

  ```
  $ make
  ```

  y copiar en el módulo **imem** las instrucciones generadas en “main.list”, respetando el formato del ejercicio 4 de la guía 1.
  Nota: verificar que se tiene instalada la toolchain de aarch64, caso contrario escribir:

  ```
  $ sudo apt install gcc-aarch64-linux-gnu
  ```

- Antes de comenzar, verificar que los registros X0 a X30 estén inicializados con los valores 0 a 30 respectivamente en la descripción del módulo **regfile**.

- Para los usuarios de Windows:
  - Instalar Ubuntu desde el app store. En WIN10 pueden seguir el tutorial: https://docs.microsoft.com/en-us/windows/wsl/install-win10
  - Luego instalar el paquete:
    ```
    $ sudo apt-get install make
    ```
  - Y:
    ```
    $ sudo apt install gcc-aarch64-linux-gnu
    ```