Laboratorio ARMv8 en SystemVerilog

Arquitectura de Computadoras 2022

Integrantes:

- Carnero, Amparo.
- Martinelli, Sofía.
- Barrios, Macarena.

Ejercicio 1

Sobre un procesador con Pipeline, dado un código de LEGv8 sin instrucciones “nop”, detectar los hazard de datos y hazard de control a la salida de imem dentro del archivo mem.dump. Una vez detectados, solucionarlos agregando instrucciones “nop” en el código y verificar que se hayan solucionado revisando nuevamente la salida de imem en mem.dump.

En primer lugar, modificamos el módulo regfile para que cuando se desee leer un registro que está siendo escrito en el mismo ciclo de clock, en lugar de escribir el contenido del registro se escriba lo que viene por la salida writeData, es decir, el contenido del registro que se está escribiendo en el módulo WriteBack.

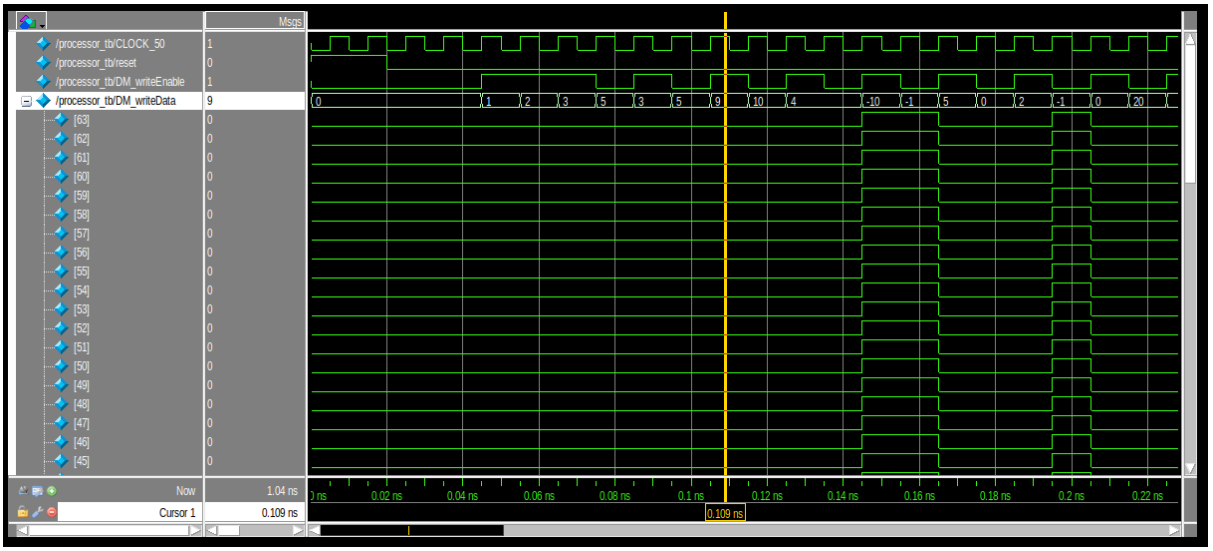
Contenido de la memoria sin agregado de instrucciones nop

Es decir, contenido de la memoria al finalizar la ejecución del código con hazard de datos y hazard de control.

```
Memoria RAM de Arm:
Address Data
0 00000000000000000000000000000000000000000000000000000000000000001
1 00000000000000000000000000000000000000000000000000000000000000010
2 00000000000000000000000000000000000000000000000000000000000000011
3 00000000000000000000000000000000000000000000000000000000000000011
4 00000000000000000000000000000000000000000000000000000000000000011
5 00000000000000000000000000000000000000000000000000000000000011001
6 00000000000000000000000000000000000000000000000000000000000000100
```

```
7 00000000000000000000000000000000000000000000000000000000000000101
8 00000000000000000000000000000000000000000000000000000000000000101
9 00000000000000000000000000000000000000000000000000000000000000010
10 00000000000000000000000000000000000000000000000000000000000010100
11 00000000000000000000000000000000000000000000000000000000000000110
12 00000000000000000000000000000000000000000000000000000000000000110
13 00000000000000000000000000000000000000000000000000000000000000111
14 00000000000000000000000000000000000000000000000000000000000001100
15 00000000000000000000000000000000000000000000000000000000000000000
16 00000000000000000000000000000000000000000000000000000000000010101
17 00000000000000000000000000000000000000000000000000000000000010101
18 00000000000000000000000000000000000000000000000000000000000011110
19 00000000000000000000000000000000000000000000000000000000000000000
20 00000000000000000000000000000000000000000000000000000000000000000
...
63 00000000000000000000000000000000000000000000000000000000000000000
```

Wave con hazard de datos



Fragmento de código a analizar:

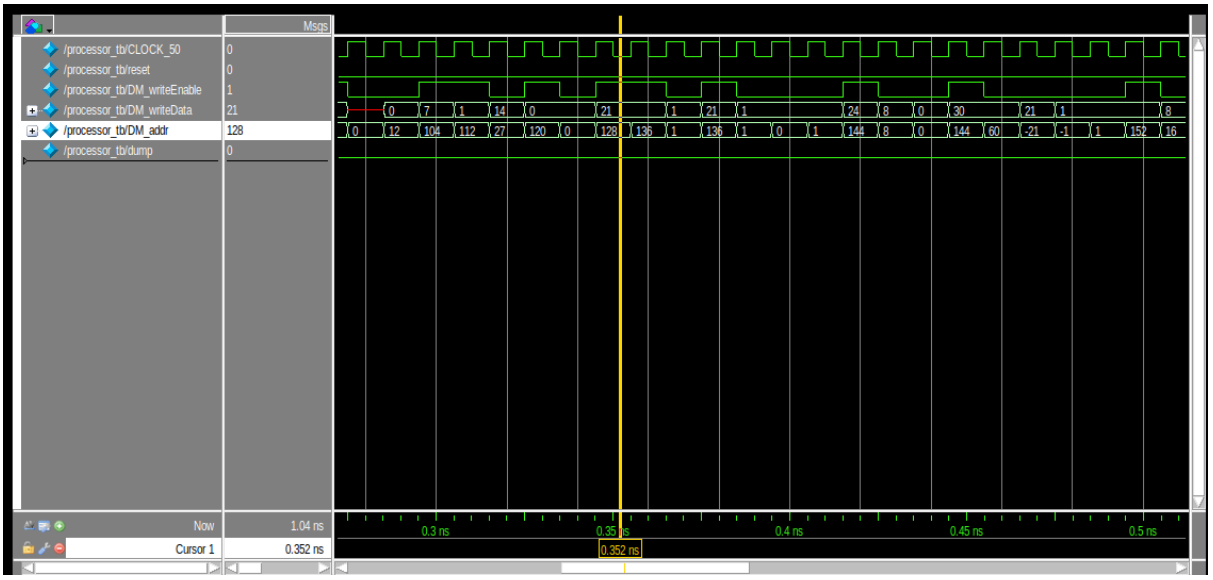
```
STUR X1, [X0, #0] // MEM 0:0x1
STUR X2, [X0, #8] // MEM 1:0x2
STUR X3, [X16, #0] // MEM 2:0x3
ADD X3, X4, X5
```

```
STUR X3, [X0, #24] // MEM 3:0x9
...
```

En esta imagen se puede observar que existe un hazard de datos a los 0.09ns, debido a que el cuarto STUR dentro del código está escribiendo el valor 3 en memoria, siendo este un valor desactualizado del registro X3. Antes del STUR se modifica X3 mediante un ADD, pero como el procesador no tiene stall ni forwarding, escribe el valor anterior del registro. Cuando writeData se actualiza a la salida del módulo WriteBack luego de la instrucción ADD X3, X4, X5 sí se escribe el valor 9 en memoria, que es el que corresponde.

Se puede observar también que la señal writeEnable y la señal de clock en ese mismo instante de tiempo están “activadas” con sus valores en ‘1’, permitiendo la escritura errónea.

Wave con hazard de control



Fragmento de código a analizar:

```
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
...
```

En este caso, se puede observar que la memoria en la dirección 15 (DM_addr = 120) se escribe correctamente. Luego, en el código se ve que viene un salto que debe ser tomado, pues el valor de x0 = 0. Pero dado que nuestro procesador es always not taken, y no cuenta con stall ni forwarding, se continúa ejecutando las instrucciones siguientes, por lo que la memoria en la dirección 16 (DM_addr = 128) y dirección 17 (DM_addr = 136) se escribe. Cuando el procesador se entera que sí debía saltar reescribe la posición 17 de memoria (DM_addr = 136) que es a donde efectivamente debía saltar.

Código del programa modificado

Código que elimina los hazard de datos y de control:

```
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
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X3, [X0, #24] // MEM 3:0x9
SUB X3, X4, X5
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X3, [X0, #32] // MEM 4:0xFFFFFFFFFFFFFFFF
SUB X4, XZR, X10
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X4, [X0, #40] // MEM 5:0xFFFFFFFFFFFFFFF6
ADD X4, X3, X4
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X4, [X0, #48] // MEM 6:0xFFFFFFFFFFFFFFF5
SUB X5, X1, X3
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X5, [X0, #56] // MEM 7:0x2
```

```
AND X5, X10, XZR
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X5, [X0, #64] // MEM 8:0x0
AND X5, X10, X3
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X5, [X0, #72] // MEM 9:0xA
AND X20, X20, X20
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X20, [X0, #80] // MEM 10:0x14
ORR X6, X11, XZR
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X6, [X0, #88] // MEM 11:0xB
ORR X6, X11, X3
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X6, [X0, #96] // MEM 12:0xFFFFFFFFFFFFFFFF
LDUR X12, [X0, #0]
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
ADD X7, X12, XZR
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
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
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X21, [X0, #128] // MEM 16:0x0 (si falla CBZ = 21)
```

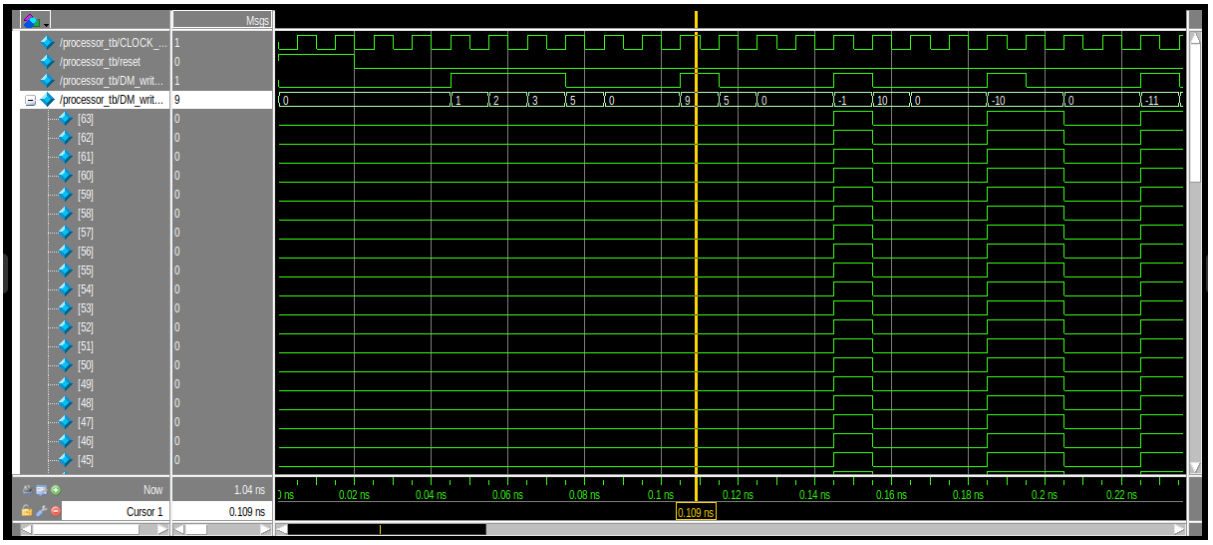
```
loop1: STUR X21, [X0, #136] // MEM 17:0x15
ADD X2, XZR, X1
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
loop2: SUB X2, X2, X1
ADD X24, XZR, X1
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X24, [X0, #144] // MEM 18:0x1 y MEM 19=0x1
ADD X0, X0, X8
CBZ X2, loop2
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X30, [X0, #144] // MEM 20:0x1E
ADD X30, X30, X30
SUB X21, XZR, X21
ADD XZR, XZR, XZR // NOP
ADD X30, X30, X20
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
LDUR X25, [X30, #-8]
ADD X30, X30, X30
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
ADD X30, X30, X16
ADD XZR, XZR, XZR // NOP
ADD XZR, XZR, XZR // NOP
STUR X25, [X30, #-8] // MEM 21:0xA (= MEM 9)
finloop: CBZ XZR, finloop
```

Contenido de la memoria al finalizar la ejecución del código modificado

```
Memoria RAM de Arm:
Address Data
0 00000000000000000000000000000000000000000000000000000000000000001
```

```
1  00000000000000000000000000000000000000000000000000000000000000000000000000000010
2  00000000000000000000000000000000000000000000000000000000000000000000000000000011
3  00000000000000000000000000000000000000000000000000000000000000000000000000001001
4  11111111111111111111111111111111111111111111111111111111111111111111111111111111
5  11111111111111111111111111111111111111111111111111111111111111111111111111110110
6  11111111111111111111111111111111111111111111111111111111111111111111111111110101
7  00000000000000000000000000000000000000000000000000000000000000000000000000000010
8  00000000000000000000000000000000000000000000000000000000000000000000000000000000
9  00000000000000000000000000000000000000000000000000000000000000000000000000001010
10 00000000000000000000000000000000000000000000000000000000000000000000000000010100
11 00000000000000000000000000000000000000000000000000000000000000000000000000001011
12 11111111111111111111111111111111111111111111111111111111111111111111111111111111
13 00000000000000000000000000000000000000000000000000000000000000000000000000000001
14 00000000000000000000000000000000000000000000000000000000000000000000000000000001
15 00000000000000000000000000000000000000000000000000000000000000000000000000000000
16 00000000000000000000000000000000000000000000000000000000000000000000000000000000
17 00000000000000000000000000000000000000000000000000000000000000000000000000010101
18 00000000000000000000000000000000000000000000000000000000000000000000000000000001
19 00000000000000000000000000000000000000000000000000000000000000000000000000000001
20 00000000000000000000000000000000000000000000000000000000000000000000000000011110
...
63 00000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Wave sin hazard de datos



Wave sin hazard de control