ITESO
Universidad Jesuita de Guadalajara

Diseño de Microprocesadores
Práctica 3: Pipeline

David Adrián Michel Torres
Eduardo Ethandrake Castillo Pulido

Realizar la implementación del procesador RISC-V pipeline visto en clase. Este procesador debe ser capaz de ejecutar un programa que calcule la multiplicación de una matriz por vector. La matriz y el vector se trasmitirá a través de una terminal serial, se recomienda el uso de docklight, pero puede cualquier otra terminal que sea capaz de transmitir valores hexadecimales directamente. La matriz tendrá una dimisión fija de 4x4 y el vector de 4x1.

Actividad a realizar

En particular en esta práctica se tiene que realizar la siguientes modificaciones o extensiones al procesador Single-Cycle:

- Segmentar el procesador Single-Cycle con los registros necesarios para poder crear un procesador pipeline con 5 etapas.
- Implementar la unidad de forwarding para eliminar o reducir el número de burbujas que se necesitan insertar cuando se presenta un hazard de datos en instrucciones de tipo aritmético-lógico.
- Implementar la unidad de data hazards para reducir el número de burbujas que se necesitan insertar para la instrucción (lw).
- Implementar predicción de saltos estática asumiendo Branch not taken para los saltos condicionales (beq y bne). La decisión de tomar el brinco o realizar el flush de registros se realizará en la etapa 4 (MEM).
- La información de la matriz y vector se pueden recibir mediante polling.

Extra:

Predicción de brincos con un bit de predicción

- Predicción de brincos con BTB y un BTH con un bit de predicción

Entregables

Un reporte en canvas que contenga:

1. Captura de pantalla donde se explique y demuestre el funcionamiento de:
    a. Forwarding Unit

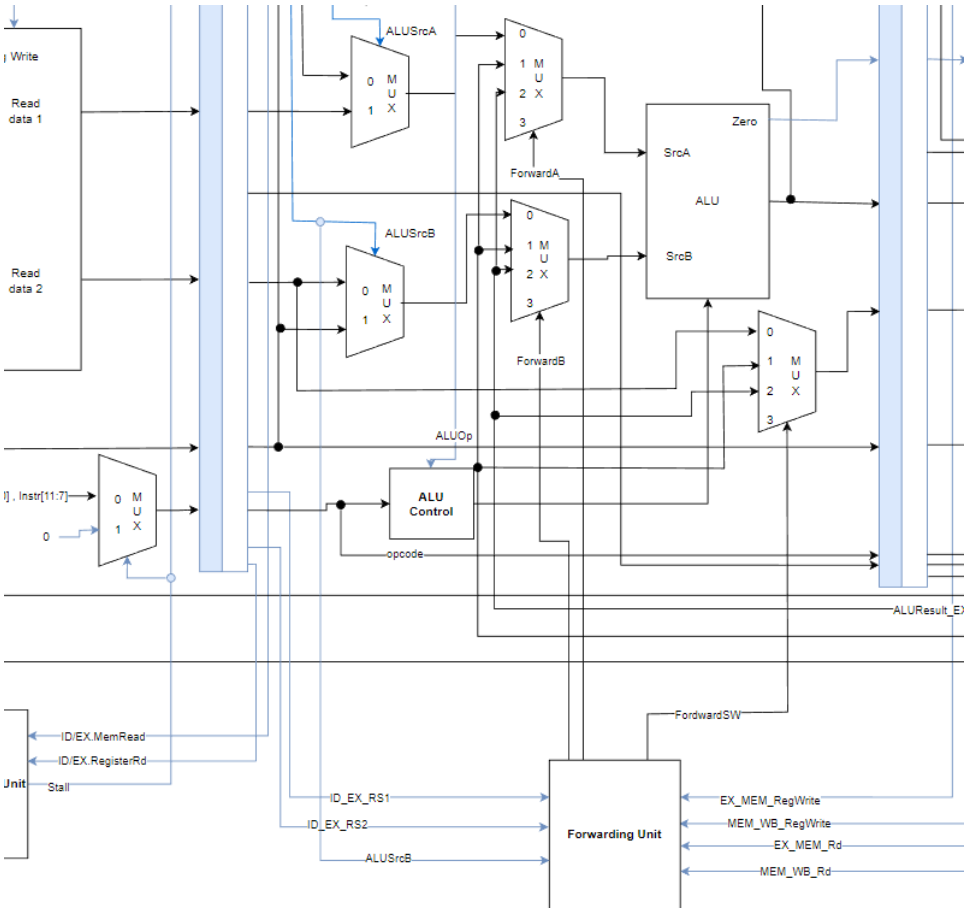

Imagen 1 – Diagrama de conexión de la forward unit

```
1  .text
2  main:
3          addi s2,s2,0x3
4          addi s5,s5,0x5
5          add s1,s2,s2  #Hazard with 1
6          sub s4,s1,s5  #Hazard with 3 & 2
7          and s6,s1,s4  #Hazard with 4 &3
8          or s8,s1,s4   #Hazard with 4
9          xor s4,s1,s5
```

Imagen 2 – Programa usado para probar dependencias de Hazard de datos

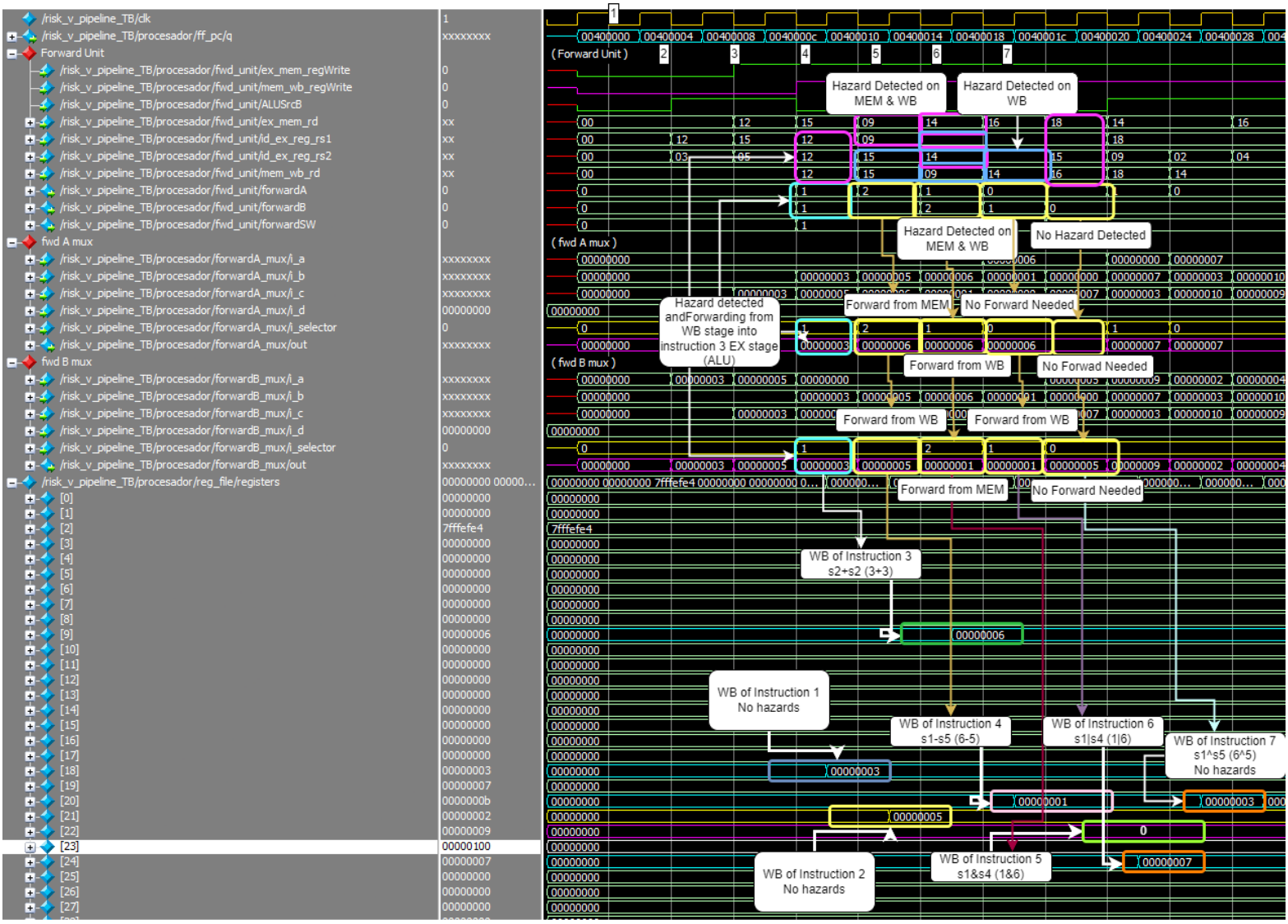

Imagen 3 - Simulación de forward unit

ITESO
Universidad Jesuita de Guadalajara

b. Hazard Detection unit para Load

```
.text
main:
1. auipc s2,    0x0000fc10 #cargar valor base de 10010004
2. addi s2, s2, 4 --- hazard with 1
3.lw s1, 4(s2) -- hazard with 2
4.add s4,s1,s5 -- hazard with 3
5.and s6,s1,s4 -- hazard with 4
6.addi s7, s7, 7
7.or s7, s7, s6 -- hazard with 6 & 5
8.add s7, s6, s7 --- hazard with 7
9.add s7, s7, s7 --- hazard with 8
10.add s6, zero, s2
11.lw s7, 4(s2)
12.sw s7, 8(s6) --- hazard with 11
13.lw s1, 4(s6)
14.lw s2, 4(s1) --- hazard with 13
```

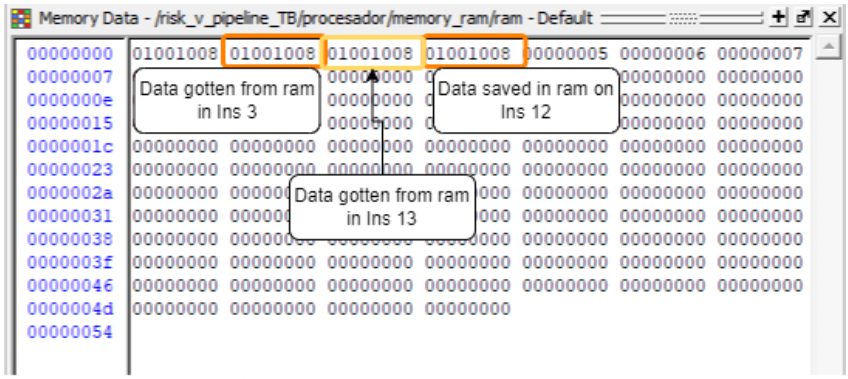

Imagen 4 - Programa & memoria RAM de la simulación de hazard detection

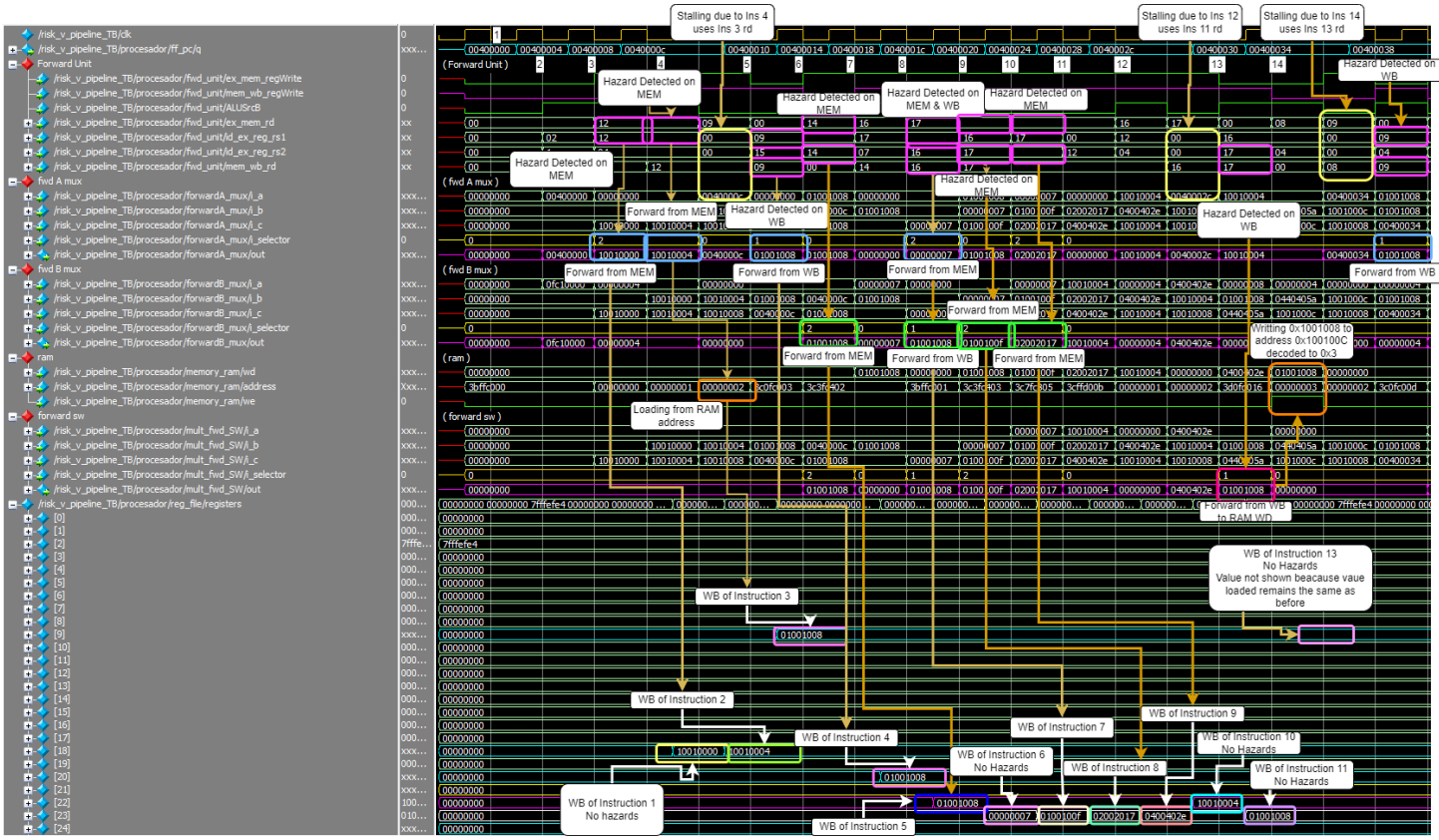

Imagen 5 – Simalación de hazard detection

c. Hazard Detection unit para beq y bne (El flush de registros en caso de una mala predicción)

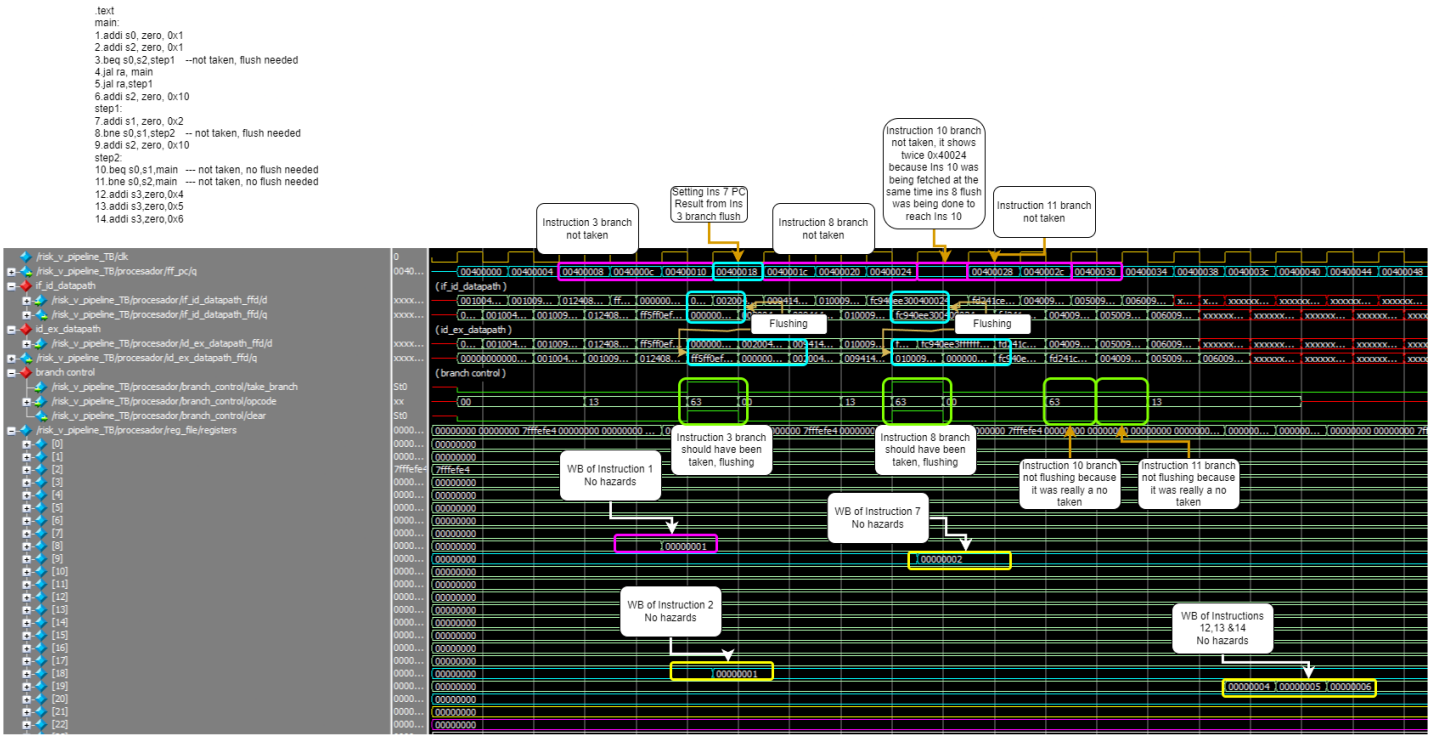

Imagen 6 – Simulación de flush de registros en caso de una mala predicción beq y bne

d. En el caso de implementar el la predicción de brincos añadir captura de pantalla de la simulación.

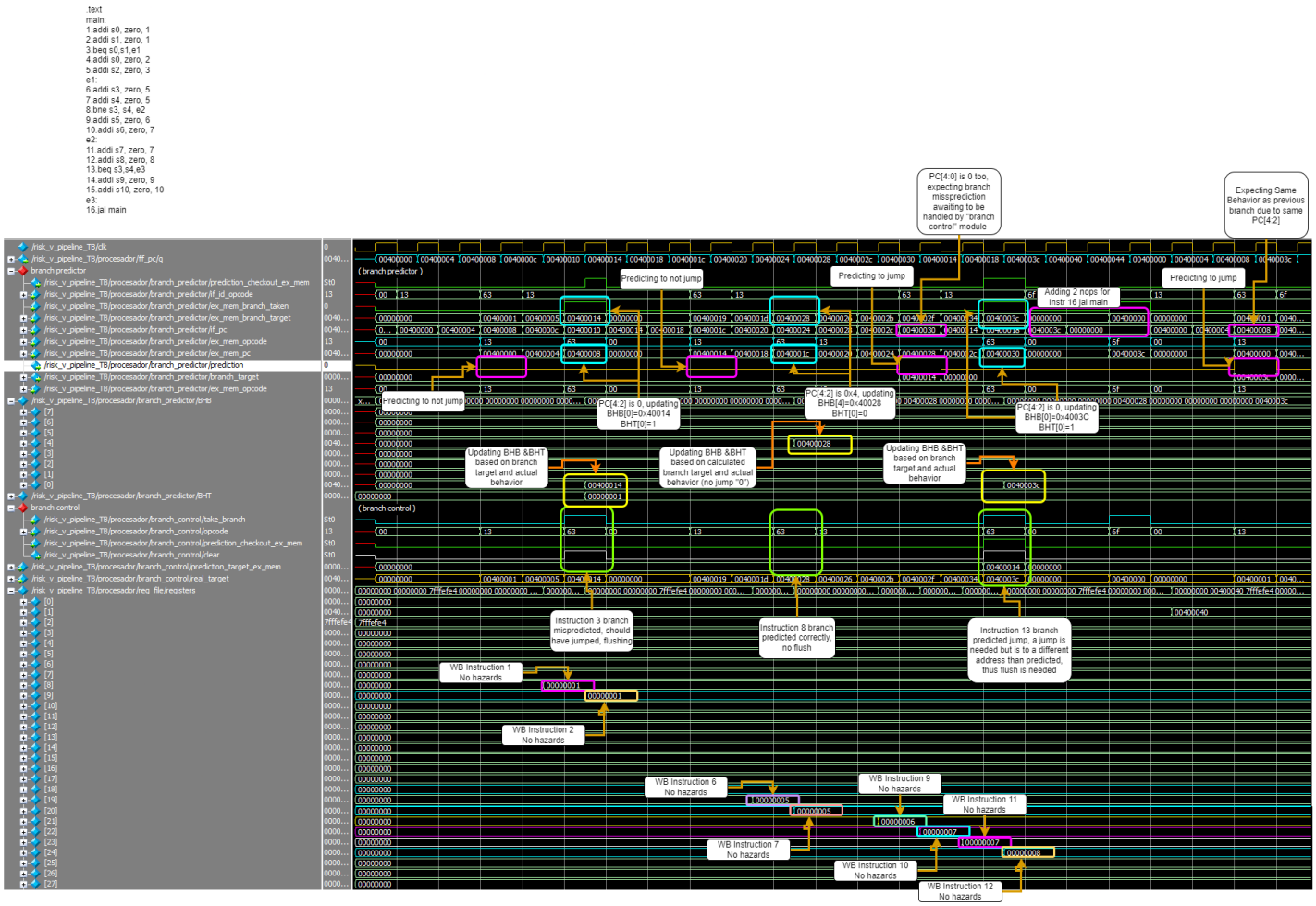

Imagen 7 – Simulación del branch predictor

e. En los puntos anteriores es necesario usar el formato de presentación de simulación.

2. Micro-arquitectura en Visio o programa equivalente.

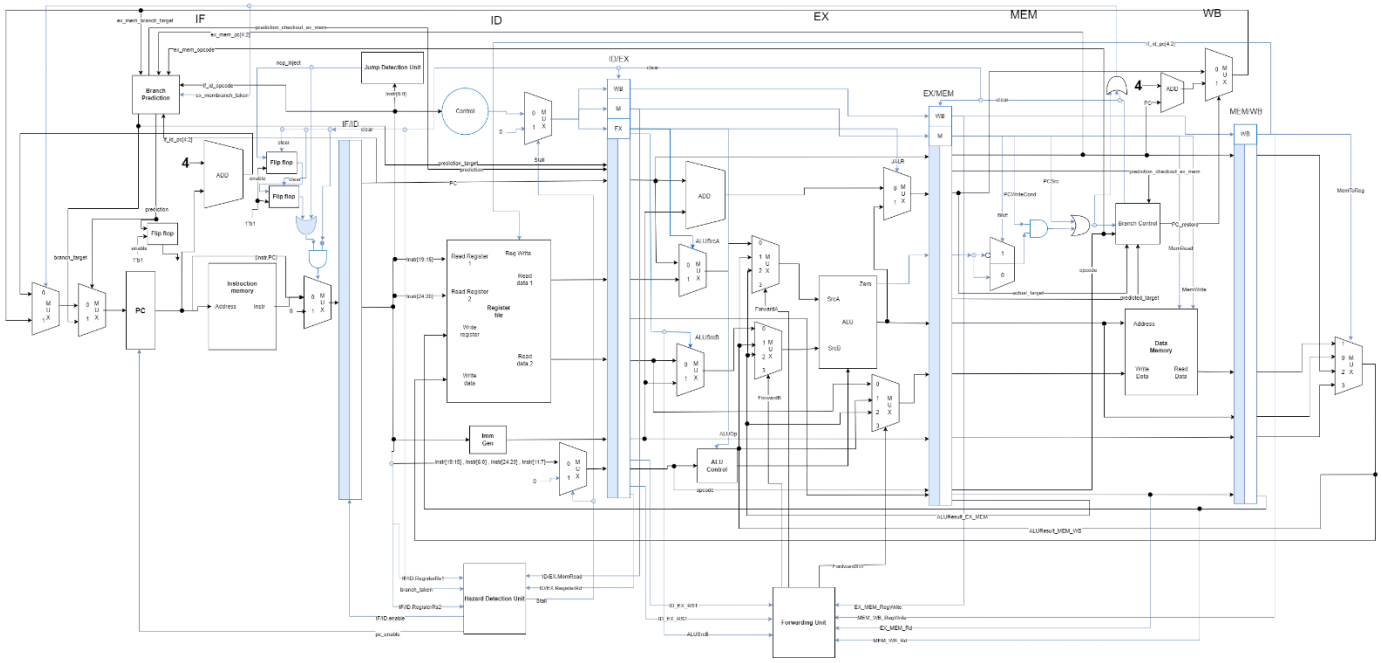

Imagen 8 – Micro arquitectura procesador Risc-V pipeline

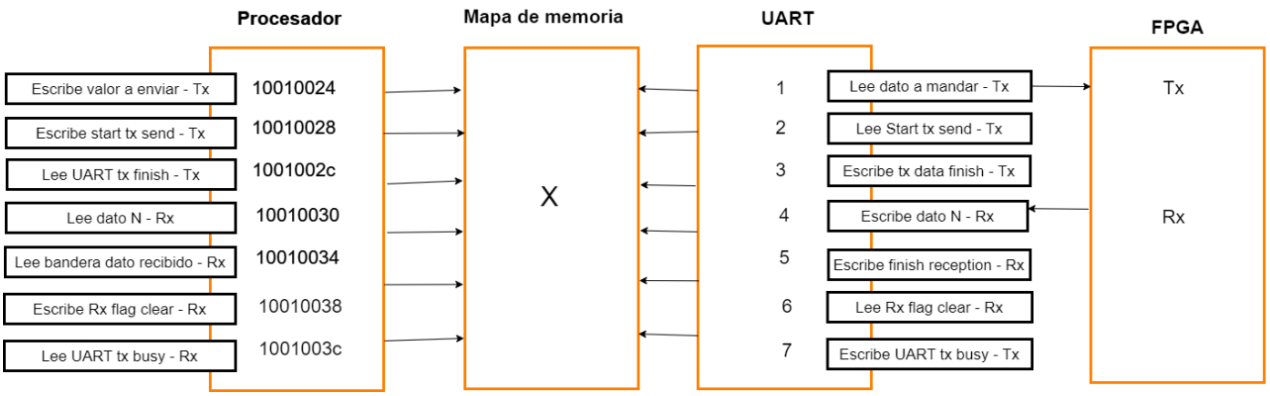

Imagen 9 – Mapeo de memoria Procesador < -- > UART

3. Captura de pantalla de las señales internas usando el Signal Tap que demuestren el funcionamiento del Forwarding Unit y Hazard Detection.

- Forwarding unit

Para demostrar el funcionamiento del forward unit, se utilizo el siguiente programa donde se buscó estresar dependencias de datos que requirieran adelantar el dato a la ejecución y tener datos correctos.

Address	Code	Basic	Source
0x00400018	0x0006aa23	sw x0,20(x13)	10: sw zero, 0x14(a3) #bajar seÃ±al p...
0x0040001c	0x00300913	addi x18,x0,3	12: addi s2,zero,0x3
0x00400020	0x00590a93	addi x21,x18,5	13: addi s5,s2,0x5
0x00400024	0x012a84b3	add x9,x21,x18	14: add s1,s5,s2
0x00400028	0x41548a33	sub x20,x9,x21	15: sub s4,s1,s5
0x0040002c	0x0144fb33	and x22,x9,x20	16: and s6,s1,s4
0x00400030	0x009a6c33	or x24,x20,x9	17: or s8,s4,s1
0x00400034	0x018b4a33	xor x20,x22,x24	18: xor s4,s6,s8
0x00400038	0x009c0993	addi x19,x24,9	19: addi s3,s8,0x9
0x0040003c	0x002a0913	addi x18,x20,2	20: addi s2,s4,0x2
0x00400040	0x004c0a13	addi x20,x24,4	21: addi s4,s8,0x4
0x00400044	0x012a8ab3	add x21,x21,x18	22: add s5,s5,s2
0x00400048	0x014aaab3	slt x21,x21,x20	23: slt s5,s5,s4
0x0040004c	0x015a69b3	or x19,x20,x21	24: or s3,s4,s5
0x00400050	0x015a8ab3	add x21,x21,x21	25: add s5,s5,s5
0x00400054	0x013a9bb3	sll x23,x21,x19	26: sll s7,s5,s3

Imagen 10 – Programa usado para demostración de forward unit usando signal tap

Una vez ejecutado el programa, se capturaron las señales de signal tap para realizar el análisis del forward unit.

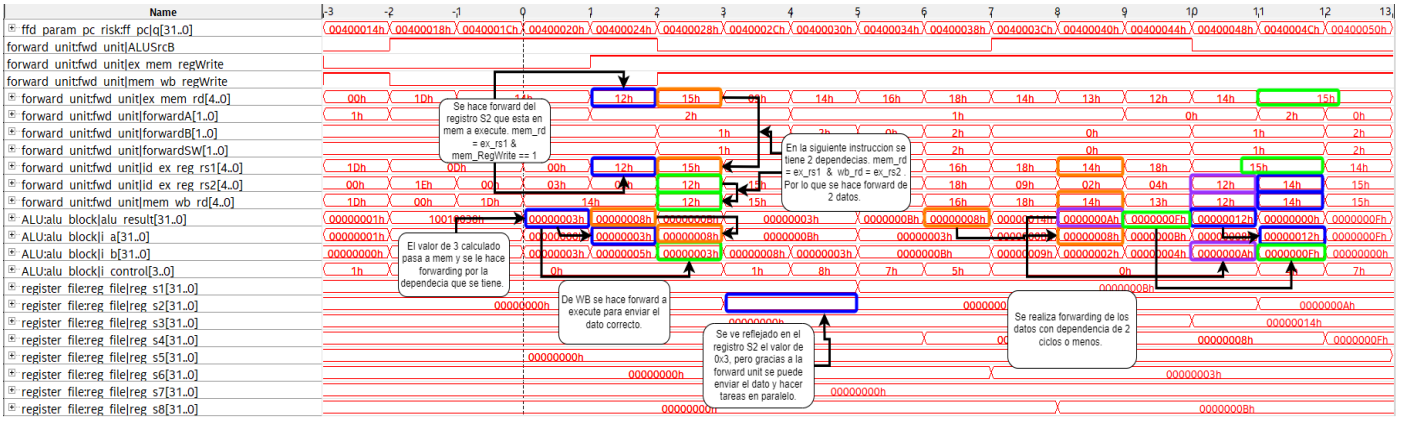

Imagen 11 – Demostración del funcionamiento de forward unit con signal tap

Se puede observar que cuando se tienen dependencias de datos, el forward unit tiene condiciones en las cuales determina cuando se debe hacer forwarding del dato. Que a resumidas cuentas es cuando es RD es igual al RS1 o RS2, dependiendo si está el dato en MEM o WB y se tengan los regWrite en valor de 0x1.

En la imagen se puede observar como se hace forwarding de los datos calculados de la ALU, que en realidad pasan al registro de MEM, y luego al registro de WB. Ahí dependiendo de le dependencia es si ese resultado se forwardea desde MEM o WB hasta la parte de ejecución como se muestra en la imagen.

- Hazard detection

Se utilizo el siguiente programa para poder validar diferentes condiciones en el Hazard detection unit.

Address	Code	Basic	Source
0x00400014	0x01e6aa23	sw x30,20(x13)	9: sw t5, 0x14(a3) #levantar seÃ‡Ã±al ...
0x00400018	0x0006aa23	sw x0,20(x13)	10: sw zero, 0x14(a3) #bajar seÃ‡Ã±al p...
0x0040001c	0x00868913	addi x18,x13,8	12: addi s2, a3, 0x8
0x00400020	0x00092483	lw x9,0(x18)	13: lw s1, 0(s2)
0x00400024	0x0096aa23	sw x9,20(x13)	14: sw s1, 0x14(a3)
0x00400028	0x0004f493	andi x9,x9,0	15: andi s1,s1,0x0
0x0040002c	0x0096aa23	sw x9,20(x13)	16: sw s1, 0x14(a3)
0x00400030	0x00892483	lw x9,8(x18)	17: lw s1, 0x8(s2)
0x00400034	0x00992a23	sw x9,20(x18)	18: sw s1, 0x14(s2)
0x00400038	0x01890b93	addi x23,x18,24	19: addi s7, s2, 0x18
0x0040003c	0x00700c13	addi x24,x0,7	20: addi s8, zero, 0x7
0x00400040	0x018ba023	sw x24,0(x23)	21: sw s8, 0(s7)
0x00400044	0x000bab83	lw x23,0(x23)	22: lw s7, 0(s7)
0x00400048	0x01890b93	addi x23,x18,24	23: addi s7, s2, 0x18
0x0040004c	0x017ba023	sw x23,0(x23)	24: sw s7, 0(s7)
0x00400050	0x000bac03	lw x24,0(x23)	25: lw s8, 0(s7)

Imagen 12 – Programa usado para demsotración de Hazard unit usando signal tap

Al correr el programa en el signal tap, se encontraron los siguientes resultados:

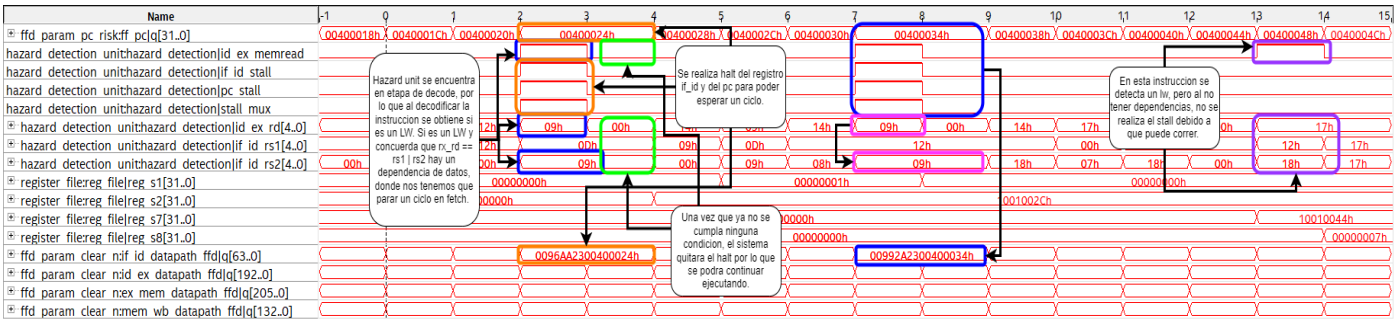

Imagen 13 – Demostración del funcionamiento de Hazard unit usando signal tap

Debido a que el hazard unit se encuentra en la etapa de decode, se puede saber si la instrucción va a ser un LW, y en dado caso que, sea una instrucción LW se valida si tiene dependencias o no. Como puede ver en la imagen los 2 primeros casos, se tenia una dependencia de ex_rd y id_rs2, el cual al ver dependencia se procede a hacer halt del FF del PC y del registro IF_ID para poder parar el sistema un ciclo, y permitir que el pipe por delante avance. En cuanto no se tenga dependencia el sistema removerá el halt para poder ejecutar la instrucción.

ITESO
Universidad Jesuita de Guadalajara

En el tercer caso, se detecto que hay un LW, pero como no tiene dependencias puede proceder sin hacer ningún halt que nos permita pararnos un ciclo. Por lo que se valida que funcione el Hazard detection unit para los LW.

4. Una tabla comparativa entre las tres organizaciones implementadas, ejecutando el factorial donde se muestre:
    a. Recursos consumidos por el FPGA
    b. Frecuencia de operación
    c. CPI (Promedio de la organización)
    d. CPU time

	MultiCycle	SingleCycle	Pipeline
Recursos consumidos FPGA	Logic utilization (in ALMs) 2,616 / 41,910 (6 %) Total registers 4064 Total pins 4 / 499 (< 1 %) Total virtual pins 0 Total DSP Blocks 2 / 112 (2 %)	Logic utilization (in ALMs) 2,201 / 41,910 (5 %) Total registers 2950 Total pins 4 / 499 (< 1 %) Total virtual pins 0 Total DSP Blocks 2 / 112 (2 %)	Logic utilization (in ALMs) 3,125 / 41,910 (7 %) Total registers 4834 Total pins 4 / 499 (< 1 %) Total virtual pins 0 Total DSP Blocks 2 / 112 (2 %)
Frecuencia de operacion	Slow 110mv 85C Fmax = 61.77 Mhz Slow 110mv 0C Fmax = 61.67 Mhz	Slow 110mv 85C Fmax = 41.87 Mhz Slow 110mv 0C Fmax = 42.37 Mhz	Slow 110mv 85C Fmax = 57.79Mhz Slow 110mv 0C Fmax = 59.63Mhz
CPI	3.8	1	1
CPU time	128ns	34ns	64ns

Tabla 1 – Comparación de recursos, frecuencia de operación, CPI, CPU time entre 3 diferentes implementaciones

5. Resultados

Una vez que se carga el programa en la FPGA, se procede a presionar el Reset para inicializar todo. Una vez inicializado el sistema, el programa realiza multiplicaciones matriz de dimensión 4x4 y vector de dimensión 4x1. El programa primero esta esperando los 16 elementos de la matriz, y después estará esperando los 4 elementos del vector para poder procesarlos y mostrar el resultado por medio de la UART.

En la imagen se puede ver una demostración de los resultados.

```
12/05/2023 01:48:06.723 [TX] - 01 02 03 04 05 06 07 08 09 0A
0B 0C 0D 0E 0F 10 01 02 03 04
12/05/2023 01:48:06.755 [RX] - 00 00 00 1E 00 00 00 46 00 00
00 6E 00 00 00 96
12/05/2023 01:48:20.685 [TX] - 01 02 03 04 05 06 07 08 09 0A
0B 0C 0D 0E 0F 10
12/05/2023 01:48:21.852 [TX] - 01 02 03 04
12/05/2023 01:48:21.859 [RX] - 00 00 00 1E 00 00 00 46 00 00
00 6E 00 00 00 96
12/05/2023 01:48:38.423 [TX] - 01 02 03 04
12/05/2023 01:48:39.117 [TX] - 01 02 03 04
12/05/2023 01:48:39.747 [TX] - 01 02 03 04
12/05/2023 01:48:40.292 [TX] - 01 02 03 04
12/05/2023 01:48:40.985 [TX] - 01 02 03 04
12/05/2023 01:48:40.995 [RX] - 00 00 00 1E 00 00 00 1E 00 00
00 1E 00 00 00 1E
```

Imagen 14 – Validación de resultados en UART – P1

En la cual se metieron los 20 datos directos, también se hizo la variante de meter 16 datos de la matriz y luego 4 datos del vector. Y también se metió la variante de enviar datos en paquetes de 4 datos 5 veces para poder llenar los 20 datos. Podemos ver en la imagen que se obtiene el resultado de la matriz como se espera.

```
12/05/2023 01:49:54.414 [TX] - 03
12/05/2023 01:49:55.075 [TX] - 03
12/05/2023 01:49:55.576 [TX] - 03
12/05/2023 01:49:56.127 [TX] - 03
12/05/2023 01:49:56.693 [TX] - 03
12/05/2023 01:49:57.196 [TX] - 03
12/05/2023 01:49:57.746 [TX] - 03
12/05/2023 01:49:58.248 [TX] - 03
12/05/2023 01:49:58.768 [TX] - 03
12/05/2023 01:49:59.304 [TX] - 03
12/05/2023 01:49:59.824 [TX] - 03
12/05/2023 01:50:00.389 [TX] - 03
12/05/2023 01:50:00.908 [TX] - 03
12/05/2023 01:50:01.434 [TX] - 03
12/05/2023 01:50:01.963 [TX] - 03
12/05/2023 01:50:02.514 [TX] - 03
12/05/2023 01:50:03.019 [TX] - 03
12/05/2023 01:50:03.522 [TX] - 03
12/05/2023 01:50:04.056 [TX] - 03
12/05/2023 01:50:04.513 [TX] - 03
12/05/2023 01:50:04.531 [RX] - 00 00 00 24 00 00 00 24 00 00
00 24 00 00 00 24
```

Imagen 15 – Validación de resultados de UART – P2

También se puede proceder manualmente a meter los 20 datos y también se obtiene los resultados de manera esperada. Por lo que se puede concluir que funcionalmente la practica funciona como se tenia esperado.

6. Historia de desarrollo en git hub usando git classroom:
    a. https://github.com/SE-O22-ITESO/p3-pipeline-david_michel-ethan_castillo

Restricciones:

Se tiene que seguir la estructura de la tarea del GPIO

Las direcciones memoria mostradas en la implementación/simulación tiene que coincidir con las mostradas en el RARS, es decir, el program counter comienza en la dirección 0x4000000 y la RAM en la dirección 0x10010000.

La frecuencia de transmisión recepción debe ser 9600 baudios

La implementación del procesador deber ser a través de un diseño estructural, no se admitirán modelos comportamentales. Pero módulos constitutivos del procesador si puede implementarse de manera comportamental, ejemplo, la ALU.

TX UART ---> GPIO[7]-->PIN_AH3

RX UART ---> GPIO[9]--->PIN_AH5

Reset ----> SW9 ----> PIN_AA30