Arquitectura de computadoras
2023

Trabajo práctico final

Implementación pipeline de un procesador MIPS simplificado en FPGA

Docentes:

Rodriguez Santiago

Pereyra Martín

Autores:

Matrícula	Apellido y Nombre
40928594	Severini Montanari, Alejo
40248013	Vega Cuevas, Silvia Jimena

Introducción

El siguiente informe tiene como objetivo abordar la implementación de un procesador segmentado de cinco etapas, un enfoque que ofrece mejoras significativas en el rendimiento en comparación de los tradicionales procesadores mono y multiciclo. Esto debido a la división y paralelización de las instrucciones en una serie de etapas clave.

En particular se analizan las siguientes etapas esenciales: IF (instruction fetch), ID (instruction decode), EX (execute), MEM (memory access) y WB (write back).

En el contexto de la arquitectura del procesador, el conjunto de instrucciones (ISA) se basa en un subconjunto de instrucciones pertenecientes al set de instrucciones MIPS IV.

Para abordar los riesgos asociados a la ejecución segmentada, el procesador cuenta con soporte para los riesgos de tipo estructural, de datos y de control. Estos riesgos se mitigan mediante la incorporación de una Unidad de Cortocircuitos y una Unidad de Detección de Riesgos.

Finalmente, se permiten dos modos de ejecución, continuo y paso a paso, en donde se tiene la opción de analizar los registros y memorias existentes del procesador.

Arquitectura

Como se mencionó anteriormente el ISA se basa en un subconjunto de instrucciones pertenecientes al set de instrucciones MIPS IV. En particular, tienen soporte las siguientes instrucciones:

Tipo R: add, sub, sll, srl, sra, sllv, srlv, srav, addu, subu, and, or, xor, nor, slt

Tipo I: lb, lh, lw, lwu, lbu, lhu, sb, sh, sw, addi, andi, ori, xori, lui, slti, beq, bne, j, jal

Tipo J: jr, jalr

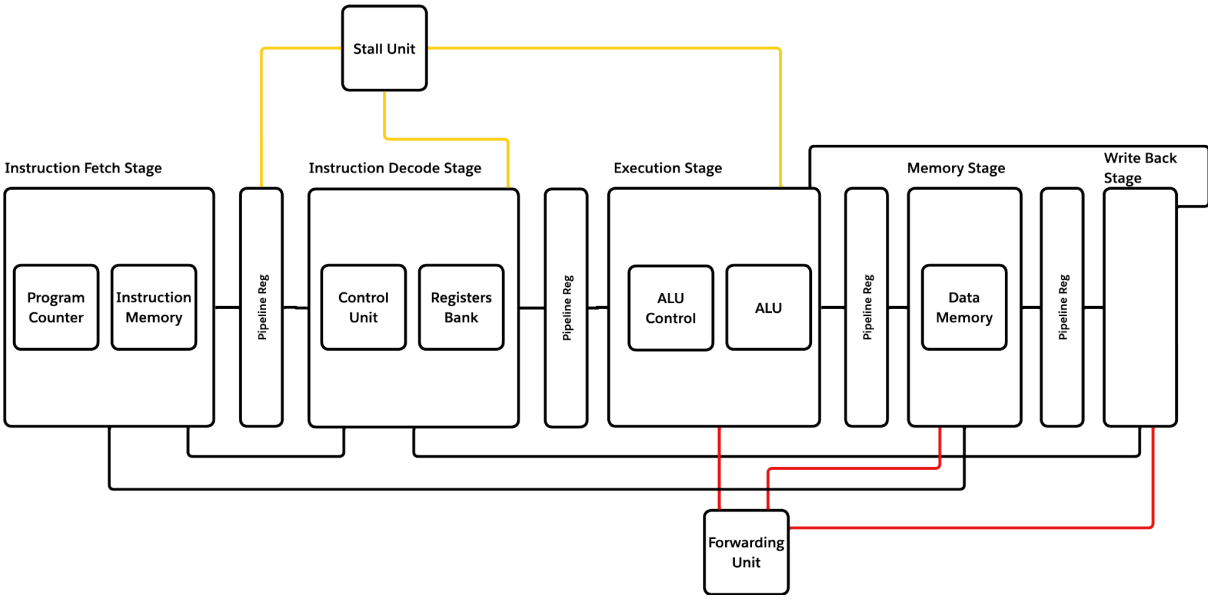
Mientras que el set de registros que utiliza es el siguiente:

0	$zero	8	$t0	16	$s0	24	$t8
1	$at	9	$t1	17	$s1	25	$t9
2	$v0	10	$t2	18	$s2	26	$k0
3	$v1	11	$t3	19	$s3	27	$k1
4	$a0	12	$t4	20	$s4	28	$gp
5	$a1	13	$t5	21	$s5	29	$sp
6	$a2	14	$t6	22	$s6	30	$fp
7	$a3	15	$t7	23	$s7	31	$ra

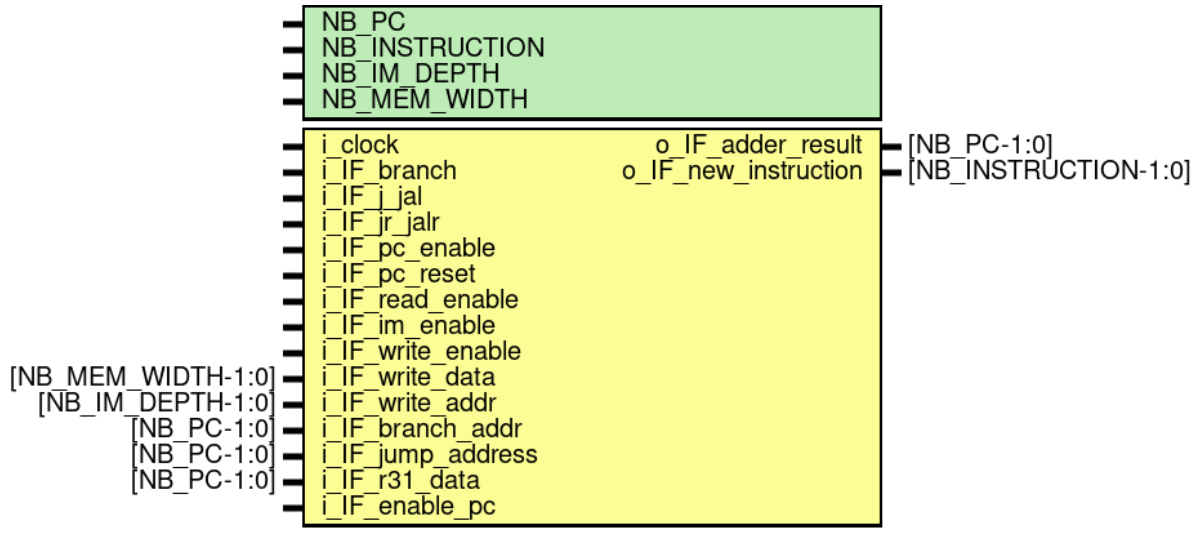
Microarquitectura

Datapath

Como se mencionó anteriormente, la arquitectura finalmente implementada es un pipeline de cinco etapas, junto con dos unidades de detección y prevención de riesgos: La stall unit y la forwarding unit.



IF - Instruction Fetch

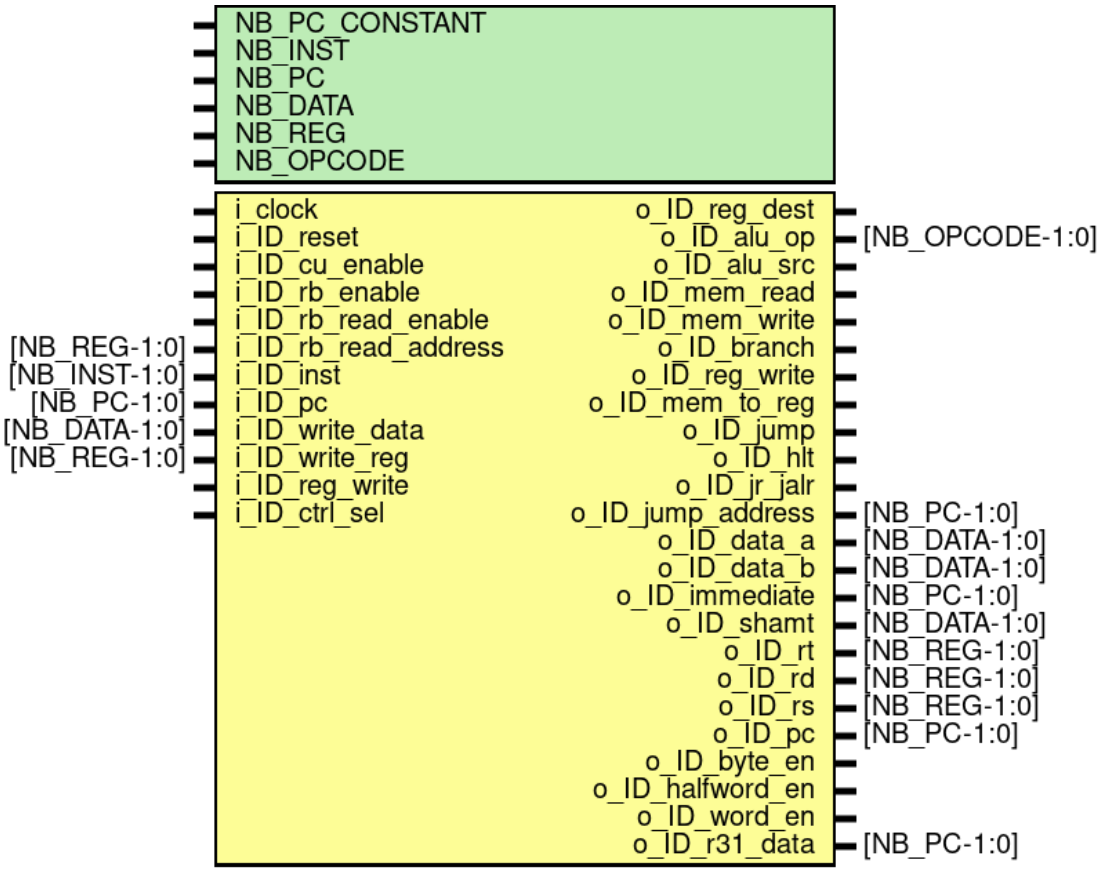


Dentro de este módulo se encuentran:

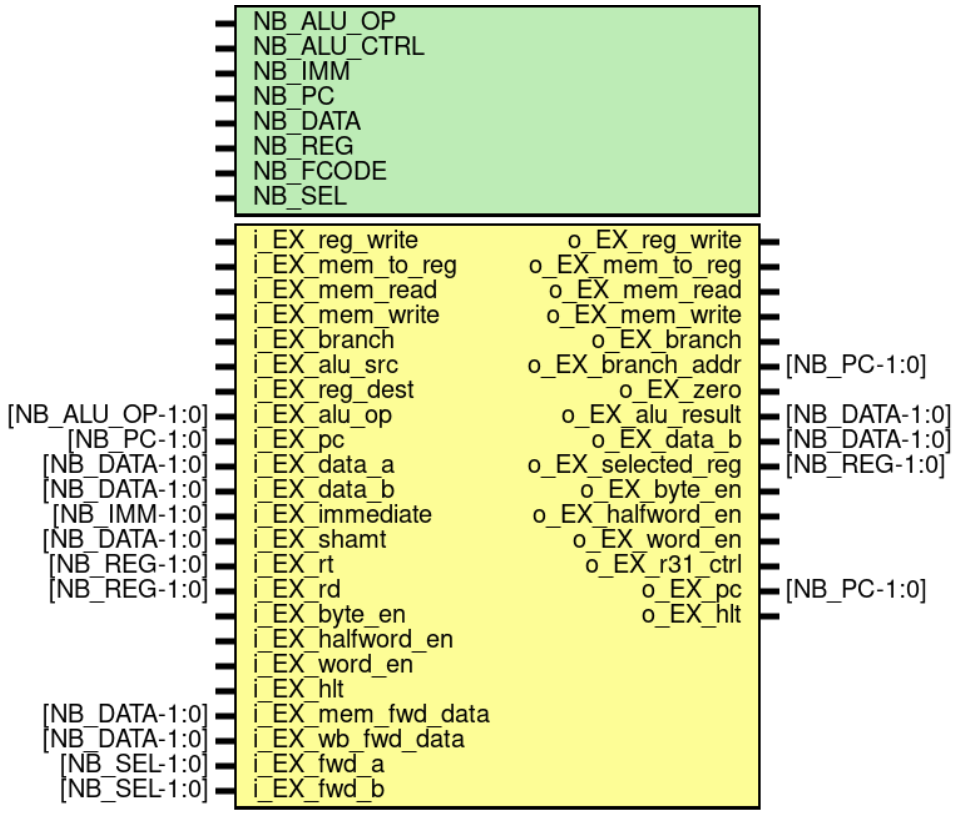
- Instruction Memory
- Program Counter
- Adder
- Mux1, Mux2 y Mux3

El Instruction Fetch es el encargado de tomar la instrucción correspondiente de la Instruction Memory y de aumentar el Program Counter correctamente (el cual en la mayoría de los casos es el que se utiliza como dirección de la IM).

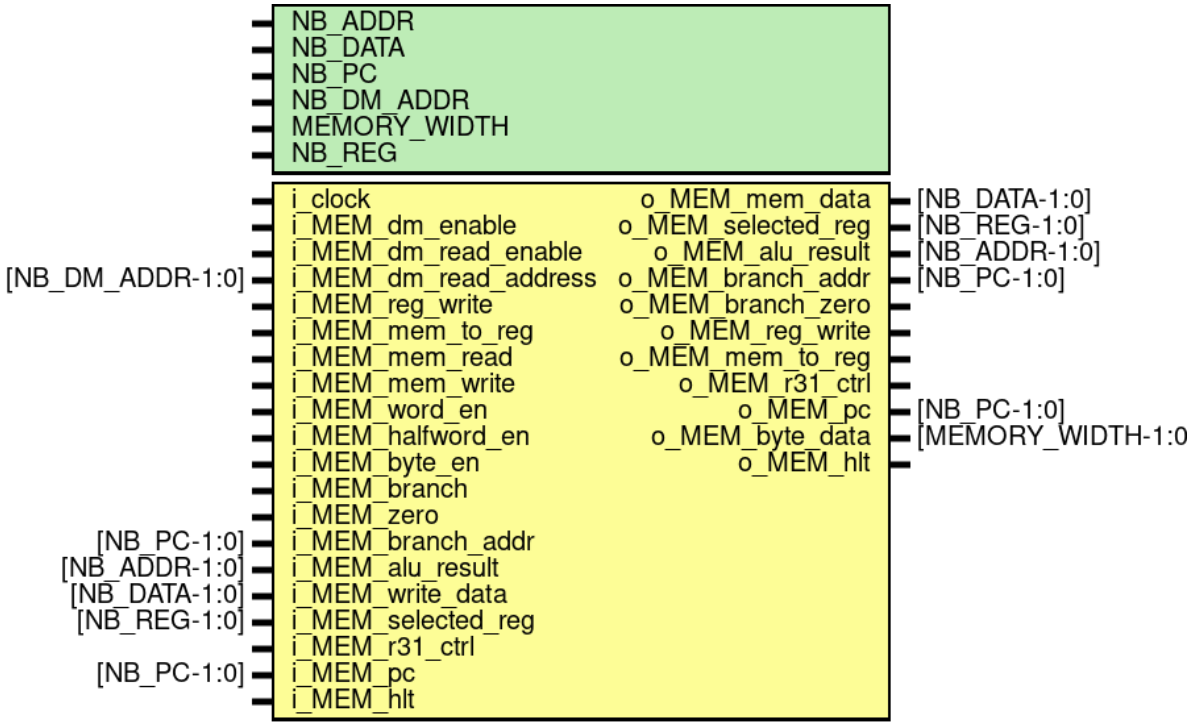
ID - Instruction Decode



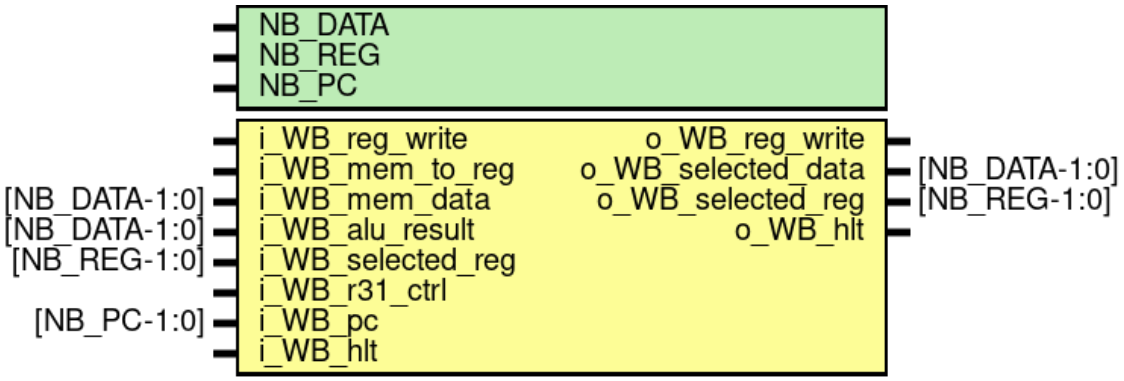
EX - Execution



MEM - Memory Access



WB - Write Back

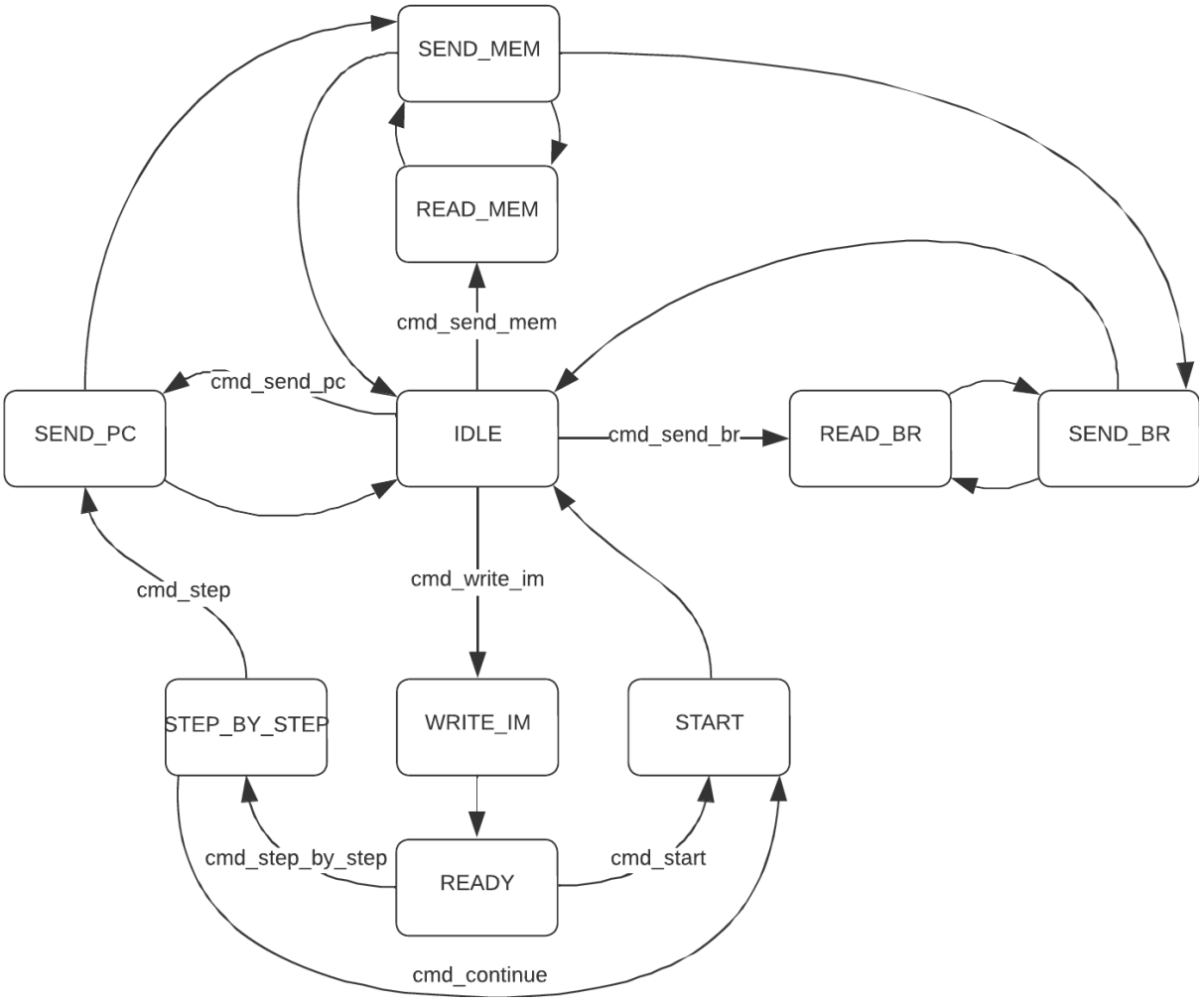


Debug unit

La debug unit es la encargada de establecer la comunicación UART y de enviar las señales correspondientes al datapath para que se pueda:

- Cargar la Instruction Memory
- Ejecutar de forma continua el programa cargado
- Ejecutar paso a paso el programa cargado
- Leer el Registers Bank
- Leer la Data Memory

Para poder realizar todo esto utiliza la siguiente máquina de estados:



Los comandos son enviados por UART utilizando una Graphical User Interface.

Escritura de IM: la escritura del programa se lleva a cabo enviando cada instrucción (4 bytes) byte por byte, de esta forma se reciben 4 bytes y se reconstruye la instrucción completa para que se pueda escribir en la Instruction Memory. Durante este proceso se

deshabilitan el Bank Register, la Data Memory, el Program Counter y la Control Unit; como así también los Pipeline Registers. De esta forma se evita que durante la escritura de la IM se traspasen datos basura en otros módulos del Data Path.

Una vez cargado el programa tenemos dos opciones: ejecución continua o paso a paso.

Ejecución continua: esto habilita todos los módulos del Data Path y lo alimenta con el clock de 50 MHz, hasta que se levante la flag de halt, por lo que vuelve al estado IDLE y se desactivan los módulos del Data Path nuevamente.

Ejecución paso a paso: de la misma manera que en la ejecución continua, habilita todos los módulos del Data Path, solo que en vez de alimentarlo con el clock de 50 MHz, lo alimenta con una señal de step, la cual se pone en alto momentáneamente cada vez que se recibe el comando Step por UART.

Luego de cada comando Step que se manda, se envían por UART el Program Counter, la Data Memory y el Bank Register.

Lectura de PC, RB y DM: para esto, se deshabilitan todos los módulos del Data Path excepto el que se quiere leer, de la misma forma que se hacía durante la escritura de la IM. Se envía byte por byte cada uno de los datos, y al finalizar, se vuelve al estado IDLE.

Detección de riesgos

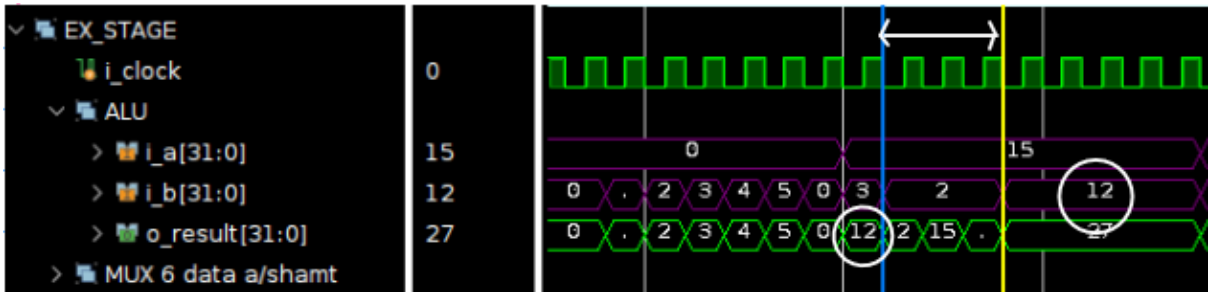
Riesgos de datos

1. RAW

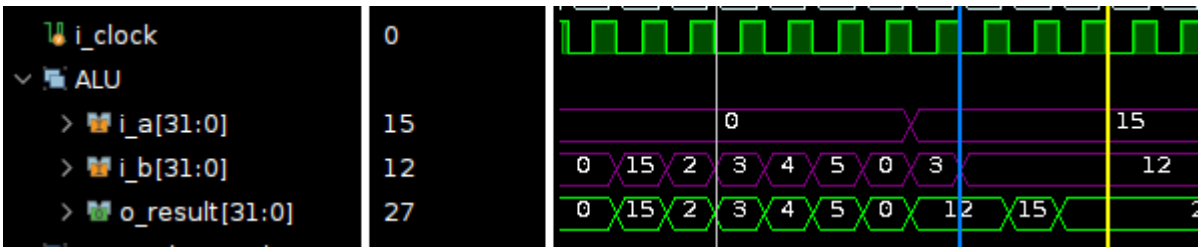
En el siguiente ejemplo se puede observar cómo se cargan los registros del $s0 al $t0 para realizar operaciones con ellos. Luego se realiza una operación de sustracción, por lo que el registro $s2 debería cambiar su valor de $s2=2 a $s2=12. Sin embargo, esto no ocurre hasta 3 ciclos después (Entre líneas amarillas y azul de la figura 3) debido a la existencia de riesgos de datos RAW. Por lo que las siguiente 3 operaciones se verán afectadas y realizarán operaciones inválidas con datos antiguos, como puede verse en la captura realizada más abajo.

```
addi $s1,$s0,15;
addi $s2,$s0,2;
addi $s3,$s0,3;
addi $s4,$s0,4;
addi $s5,$s0,5;
addi $t0,$s0,0;

sub $s2,$s1,$s3;
and $s0,$s1,$s2;
or $s5,$s1,$s2;
add $t0,$s1,$s2;
```

$s0	0
$s1	15
$s2	2
$s3	3
$s4	4
$s5	5
$t0	0



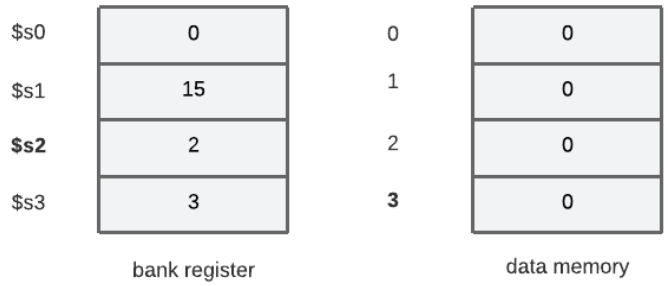
Con forwarding unit agregada, puede verse que inmediatamente luego de la operación de sustracción se utiliza el valor correspondiente a $s2, debido a que se cortocircuitó el valor a almacenar en $s2.

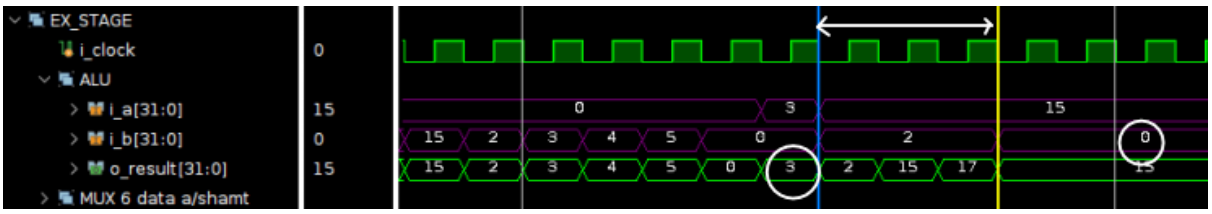


2. RAW con Load

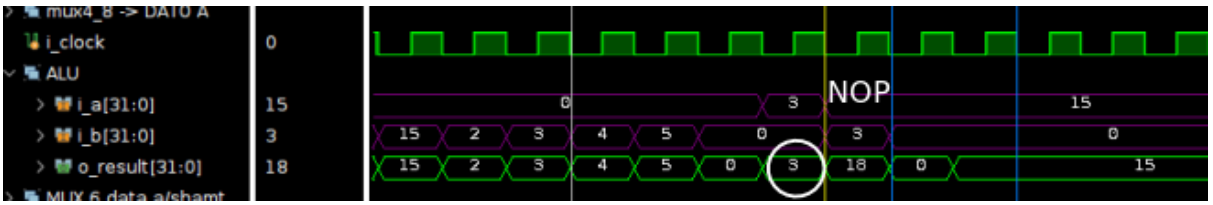
En este caso, los registros se cargaron de la misma manera que en el ejemplo anterior, con la diferencia de que ahora se utiliza una operación load en lugar de una de sustracción. Debido al riesgo de datos existente también se puede apreciar que el registro $s2 tardará en actualizarse 3 ciclos para pasar de $s2=2 a $s2=0.

```
addi $s1,$s0,15;
addi $s2,$s0,2;
addi $s3,$s0,3;
addi $s4,$s0,4;
addi $s5,$s0,5;
addi $t0,$s0,0;

lw $s2,0($s3);
and $s0,$s1,$s2;
or $s5,$s1,$s2;
add $t0,$s1,$s2;
```





Tras agregar la stall unit se obtiene:



Donde se puede ver que tras la instrucción nop el registro $s2 logra actualizarse con el valor 0 utilizando la forwarding unit en los dos primeros ciclos siguientes al nop (entre lineas azules).

Riesgos de control

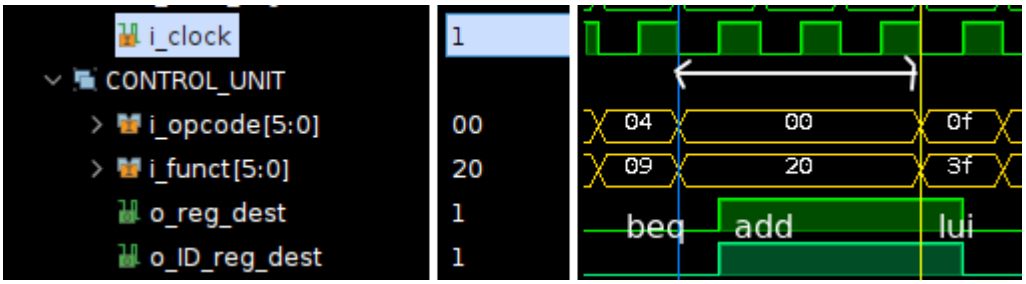
Riesgos con branches

Para este ejemplo se buscó que tanto $s1 y $s2 sean iguales para que cuando se aplique la operación beq $s1,$s2,9 el salto sea tomado, siendo su destino la instrucción lui $t2,255. Sin embargo, en la imágen de referencia puede verse como pueden entrar 3 instrucciones adicionales al pipeline antes de que se logre implementar el salto, esto es el riesgo de control.

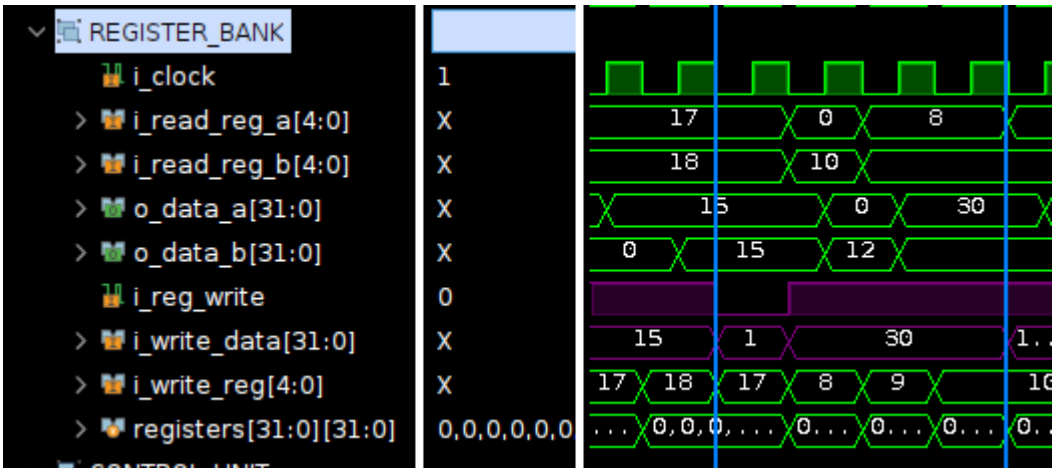
```
addi $t0,$s0,10;
addi $t1,$s0,11;
addi $t2,$s0,12;
addi $t3,$s0,13;
addi $t4,$s0,14;

addi $s1,$s0,15;
addi $s2,$s0,15;

beq $s1,$s2,9;
add $t0,$s1,$s2;
add $t1,$s1,$s2;
add $t2,$s1,$s2;
or $t3,$s1,$s2;
sub $t4,$s1,$s2;
or $s5,$s1,$s2;
sub $t1,$s1,$s2;
or $s4,$s1,$s2;
sll $s0,$s1,3;
sll $s0,$s1,3;
lui $t2,255;
```

$s0	0
$s1	15
$s2	15
$s3	3
	...
$t0	10
$t1	11
$t2	12
$t3	13
$t4	14

bank register



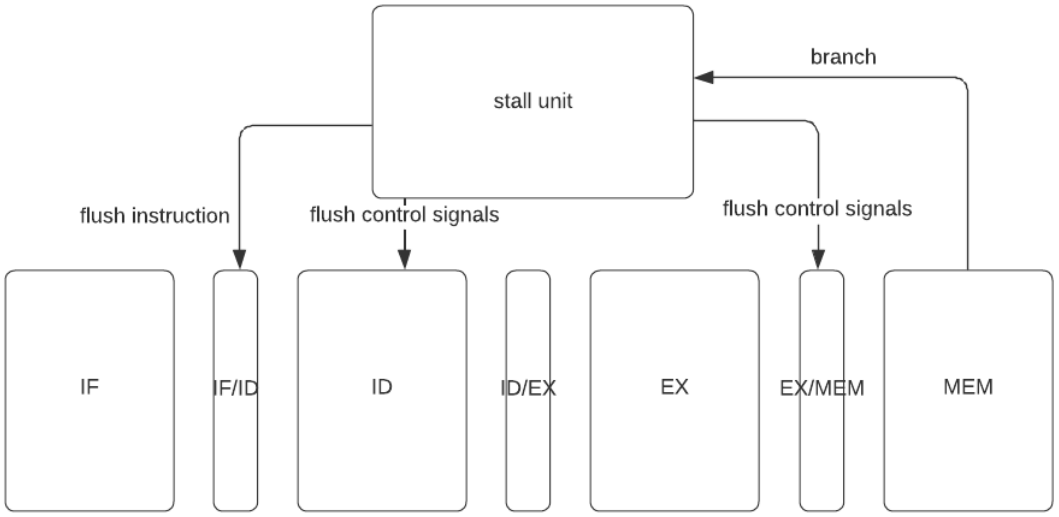
En la siguiente imagen puede verse que debido a las tres instrucciones siguientes al branch, son escritos innecesariamente los registros $t0,$t1 y $t2 (8,9 y 10 en la imagen respectivamente) con el valor 30.



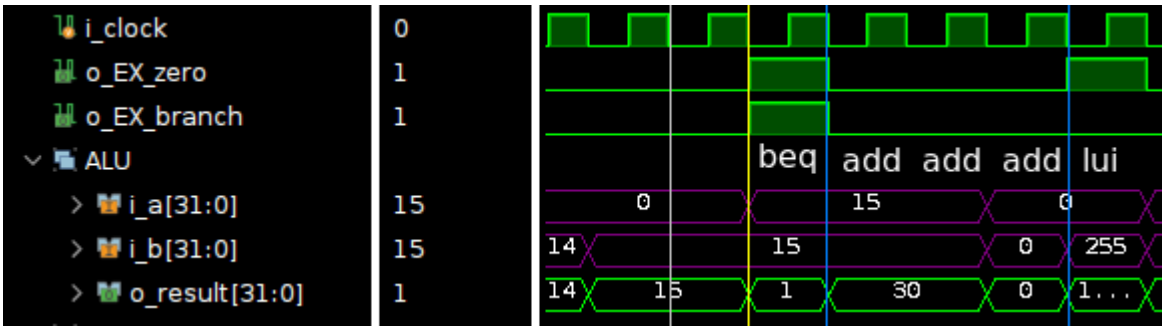
Solución

En este caso, se optó por mitigar el riesgo de control implementando una estrategia más eficiente para evitar los stalls causados por el tiempo de espera necesarios (tres clocks) para la determinación del estado del branch. En lugar de esperar a conocer el resultado, se decidió asumir que el branch no se toma. Esto permite que tres instrucciones adicionales ingresen al pipeline antes de que se determine en la etapa de MEM si el branch es tomado o no.

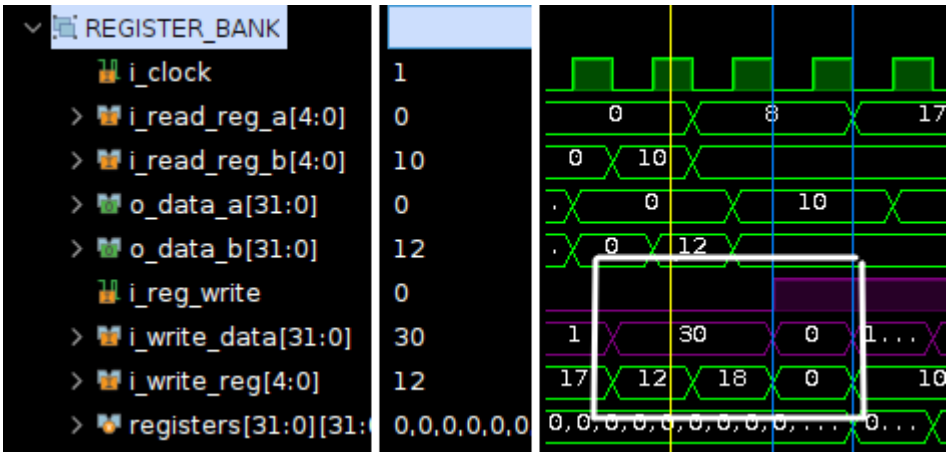
Si se confirma que el branch realmente se toma, se activará la **stall unit**, la cual enviará una señal de "flush" a las etapas de fetch, decode y execute. Esta acción evita que estas instrucciones modifiquen los valores almacenados en las memorias o registros, tal como se muestra en la figura adjunta.



En la captura proporcionada, se puede observar que las dos instrucciones siguientes al "beq" aparentemente no presentan ningún cambio. Sin embargo, es evidente que la tercera instrucción "add" está realizando operaciones con ambos operandos igualados a cero. Esto se debe a que el "flush" en el registro del pipeline "IF/ID" provocó que la instrucción "add" se transformara en "sll $zero,$zero,0", una instrucción que no genera ningún cambio o efecto.



El cambio más significativo puede observarse en el módulo **bank register**, donde vemos que en los registros 12 ($t4) y 18 ($t3) no se escribe nada y en el registro 0 se escribe 0 por lo que tampoco hay cambio alguno.



Riesgos de control en jumps

Dado que cuando se utiliza una instrucción de tipo jump, hasta que esta es ejecutada guardando su valor en el program counter, se pueden meter hasta dos instrucciones adicionales en el pipeline. Esto implicaría que hay riesgos de control para este tipo de instrucciones.

Para esto se implementó una detección de flag de jump en la stall unit. Tras ser detectada se mandan señales para limpiar (flush) las instrucciones entrantes que no deberían provocar cambio alguno en las memorias.

Análisis de timing

Sabiendo que el tiempo de setup es el intervalo antes de la transición activa del clock durante el cual los datos deben mantenerse y que el tiempo de hold es el intervalo luego de la transición activa, ambos fueron tenidos en cuenta a la hora de analizar cómo se comportaba el procesador a diferentes frecuencias.

Analizando los reportes de Vivado, cuando el Worst Negative Slack es negativo, significa que el delay del path combinacional más largo es mayor que el período del clock. Por lo que deberán tomarse medidas como disminuir la frecuencia del clock para así aumentar el periodo o bien disminuir el clock skew.
En nuestro caso, comienza a ser violado tras subir de 90MHz.

50 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.611 ns	Worst Hold Slack (WHS):	0.119 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3935	Total Number of Endpoints:	3935	Total Number of Endpoints:	1088

All user specified timing constraints are met.

60 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.611 ns	Worst Hold Slack (WHS):	0.119 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3935	Total Number of Endpoints:	3935	Total Number of Endpoints:	1088

All user specified timing constraints are met.

70 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.667 ns	Worst Hold Slack (WHS):	0.042 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3935	Total Number of Endpoints:	3935	Total Number of Endpoints:	1088

All user specified timing constraints are met.

80 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.169 ns	Worst Hold Slack (WHS):	0.112 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3937	Total Number of Endpoints:	3937	Total Number of Endpoints:	1089

All user specified timing constraints are met.

90 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0.016 ns	Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	-0.031 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	3	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3937	Total Number of Endpoints:	3937	Total Number of Endpoints:	1089

Timing constraints are not met.

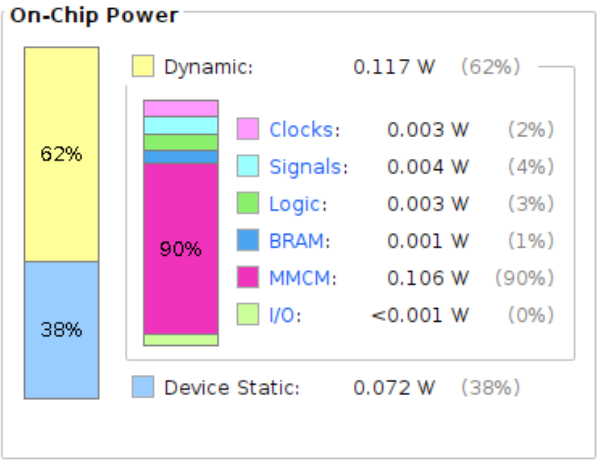
Análisis de potencia

Los siguientes resultados pueden verse con la frecuencia de clock finalmente elegida para el procesador de 50 MHz.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

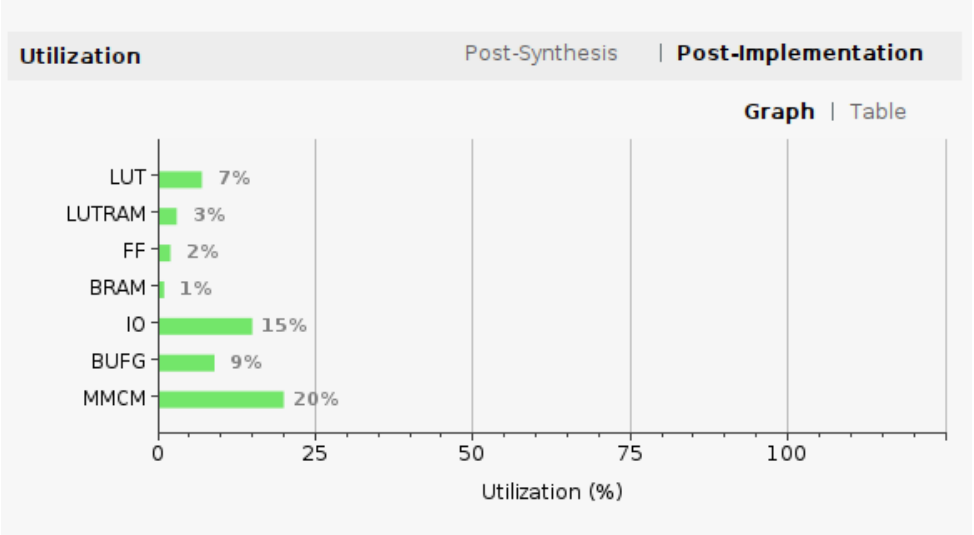
Total On-Chip Power:	**0.189 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.9°C**
Thermal Margin:	59.1°C (11.7 W)
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



Recursos utilizados en el proyecto

Tras la implementación se tiene que el primer recurso más utilizado es el Mixed-Mode Clock Manager (MMCM) utilizado para generar múltiples clocks



Conclusión

En este trabajo, implementamos un procesador de 5 etapas que efectivamente previene riesgos de datos, control y estructurales.
Utilizando un subconjunto del set de instrucciones MIPS IV se lograron ejecutar instrucciones de tipo R, I y J. Donde las R involucran instrucciones aritméticas, las I operaciones lógicas y las J instrucciones de salto.
Finalmente, para evaluar la performance y consumo de potencia de nuestro diseño, se utilizaron las herramientas provistas por Vivado. Basado en los resultados, recomendamos operar este procesador con una frecuencia menor a 80 MHz, tomando finalmente una frecuencia de 50 MHz para obtener un margen de error mayor.