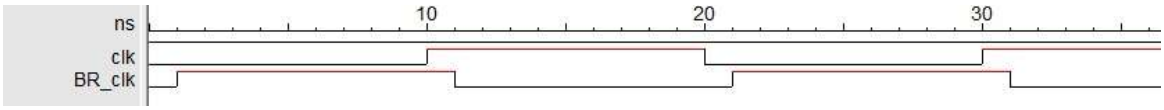
Proyecto Optativo 1: Gestión de riesgo de datos

Fernando Aliaga Ramón - 610610

Paso 1: Escritura en el BR en el flanco de bajada

Para tener un banco de registros que pueda escribir y leer en el mismo ciclo debemos modificar su señal de reloj. Dicha señal ahora atraviesa un inversor, por lo que sus valores serán así:



Como podemos ver, las señales son contrarias con el retardo de 1ns que genera la puerta *not* del banco de registros.

El funcionamiento del banco de registros, como ya vimos en la primera práctica, depende del flanco ascendiente del reloj, es decir, cuando la señal pasa de 0 a 1. La operación de escritura debe estar preparada antes del flanco, mientras que la operación de lectura obtiene una señal sin fallos transcurrido un tiempo después del flanco. Dichos tiempos son el tiempo de *setup* y el de *delay*:

$$T_{delay} = T_{delayReg8} + T_{delayMux4\_1}$$

$$T_{delay} = T_{delayReg8} + T_{delayMux8\_2\ (1)} + T_{delayMux8\_2\ (2)}$$

$$T_{delay} = 1ns + 2ns + 2ns = 5ns$$

$$T_{setup} = T_{delayDec2\_4} + T_{delayMux8\_2} + T_{setupReg8}$$

$$T_{setup} = 2ns + 3ns + 1ns = 6ns$$

Si realizamos la modificación de poner un inversor de la señal del reloj dentro del banco de registros, podremos realizar primero una escritura y después una lectura. Entre estas dos operaciones se encontrará el flanco ascendente del reloj del banco, el cual llegará con 1ns de retraso respecto del flanco descendente del reloj del circuito.

El camino que deberá recorrer la instrucción de escritura será el siguiente:

$$T_{lectura} = T_{reg8} + T_{nand3} + T_{and} + T_{dec} + T_{setupReg8} =$$

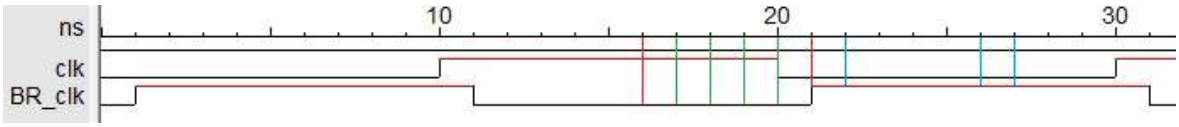
$$1ns + 1ns + 1ns + 1ns + 1ns = 5ns$$

El camino que deberá recorrer la instrucción de lectura será el siguiente:

$$T_{lectura} = T_{reg8} + T_{mux} + T_{setupReg8} =$$

$$1ns + 4ns + 1ns = 6ns$$

Por lo tanto, el tiempo que debe estar el reloj del banco de registros a 0 es de 5ns y el tiempo que debe estar a 1 será 6ns.



Paso 2: Anticipación de operandos o cortocircuitos

La anticipación de operandos permite usar un operando (registros 0-3) en el momento en el que se obtiene, sin necesidad de esperar a la etapa WB. Para ello conectaremos los registros A y B con la salida del *Adder*, la salida de la memoria y la salida de las lecturas del banco de registros. Cuando comience la etapa de ejecución (EX), deberemos elegir entre los datos que podemos usar. Para realizar dicha elección, basta con definir una función lógica con los datos de entrada que controle el multiplexor de la entrada del *Adder*:

Datos de entrada
RA (2bits)
RB (2bits)
REG_EX/MEM (2bits)
WB_EX/MEM (1bit)
NOP_EX/MEM (1bit)
REG_MEM/WB (2bits)
WB_ MEM/WB (1bit)
NOP_ MEM/WB (1bit)

La función lógica será la siguiente:

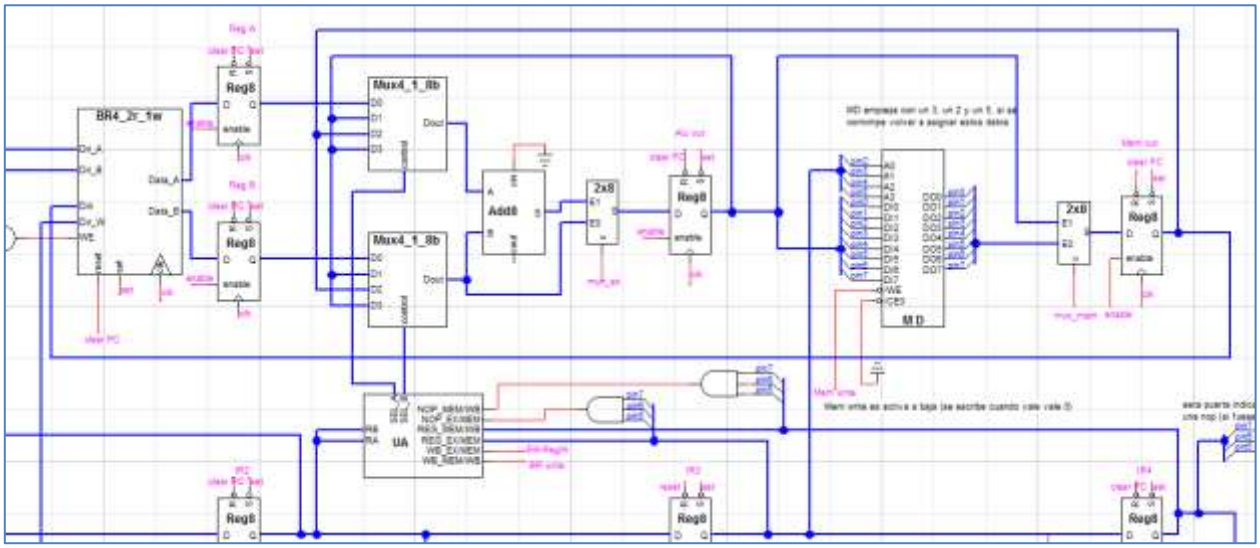
si (RA≠REG_EM | RA≠REG_MW | WB_EM=0 | WB_MW=0 | NOP_EM=1 | NOP_MW=1)
entonces SelA = 00

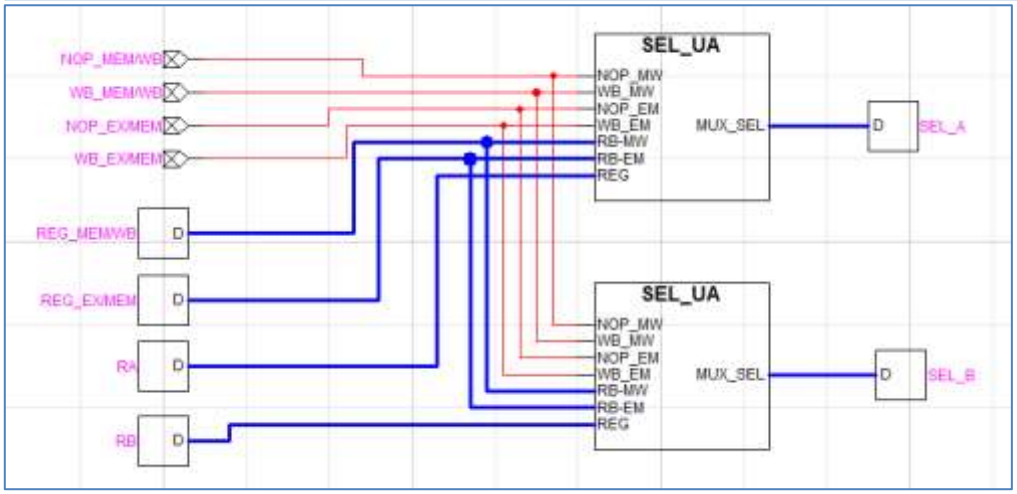
si (RA=REG_EX/MEM & WB_EX/MEM = 1 & NOP_EX/MEM = 0)
entonces SelA = 01

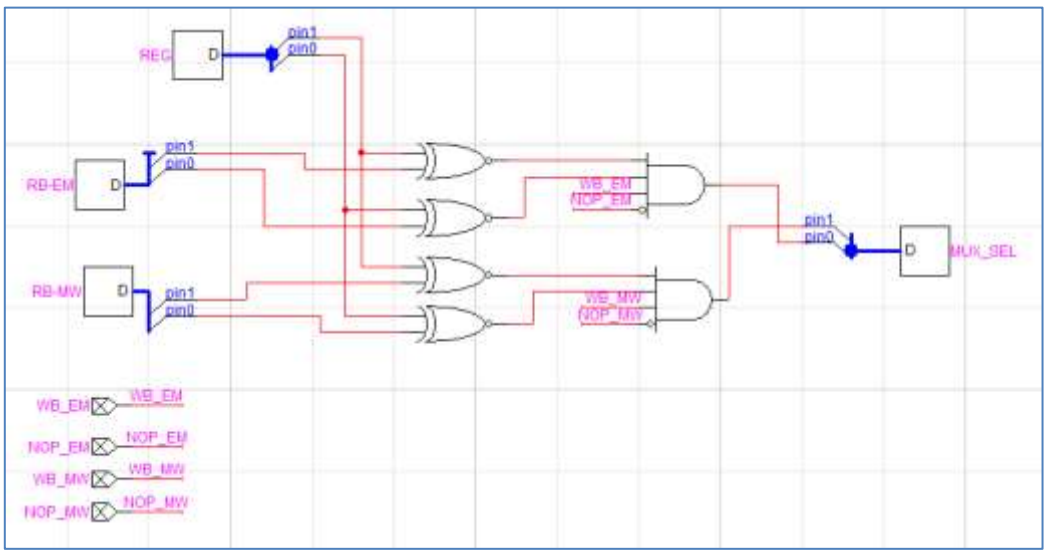
si (RA=REG_MEM/WB & WB_MEM/WB = 1 & NOP_MEM/WB = 0)
entonces SelA = 10

si (RA=REG_EX/MEM & WB_EX/MEM = 1 & NOP_EX/MEM = 0) &
(RA=REG_MEM/WB & WB_MEM/WB = 1 & NOP_MEM/WB = 0)
entonces SelA = 11

El hardware necesario para implementar los cortocircuitos consta de una Unidad de Anticipación (UA) y de un multiplexor a la entrada del *Adder*.







Viendo el diseño podemos eliminar ciertos riesgos de datos:

Operación Load

Op	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9
LD	IF	ID	EX	**MEM**	BW				
LD		IF	ID	EX	MEM	BW			
ST			IF	ID	**EX**	MEM	BW		
JMP		IF	ID	EX	MEM	BW			
ADD			IF	ID	**EX**	MEM	BW		

Operación Add

Op	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9
ADD	IF	ID	**EX**	MEM	BW				
LD		IF	ID	**EX**	MEM	BW			
ST		IF	ID	**EX**	MEM	BW			
JMP		IF	ID	EX	MEM	BW			
ADD		IF	ID	**EX**	MEM	BW			

Al añadir nuevos componentes que operan en la etapa EX, deberemos realizar un nuevo estudio del camino crítico que marcará el tiempo del reloj.

T_{EX} = 1ns(Reg8) + 1ns(and3&xnor) + 1ns(and4) + 4ns(mux) + 16ns(Adder) + 2ns(mux) +1ns($T_{setupReg8}$) = 26ns

Por lo tanto, deberemos redefinir el tiempo del reloj a ciclos de 26ns para el correcto funcionamiento del procesador.

Paso 3: Detención del segmentado

La anticipación de operandos nos permitía eliminar muchos riesgos de datos, sin embargo existía riesgo tras una operación de tipo Load si queríamos leer el registro que primero escribiría la operación Load. Para solucionar estos riesgos solo era posible a nivel de código, incluyendo una operación entre medio, ya sea una *nop* u otra instrucción. Para abstraer de esa tarea al programador del código se crea la *Unidad de Detención*, que se encarga de parar la ejecución hasta que la instrucción con riesgo obtenga sus operandos.

Dicha *Unidad* tendrá varias entradas y una única salida, la cual indica si vale 1 que hay que detener la ejecución y 0 si no hay que detener nada:

Datos de entrada
RA (2bits)
RB (2bits)
REG_ID/EX (2bits)
OP_ID/EX (2bit)
BRWrite_ID/EX (1bit)

Con estos datos de entrada podemos definir una función lógica que devuelva una correcta señal de *STOP*:

Si(BRWrite=1 & OP=00 & RA=$REG_{ID/EX}$) o (BRWrite=1 & OP=00 & RB=$REG_{ID/EX}$) entonces STOP=1

Sino STOP = 0

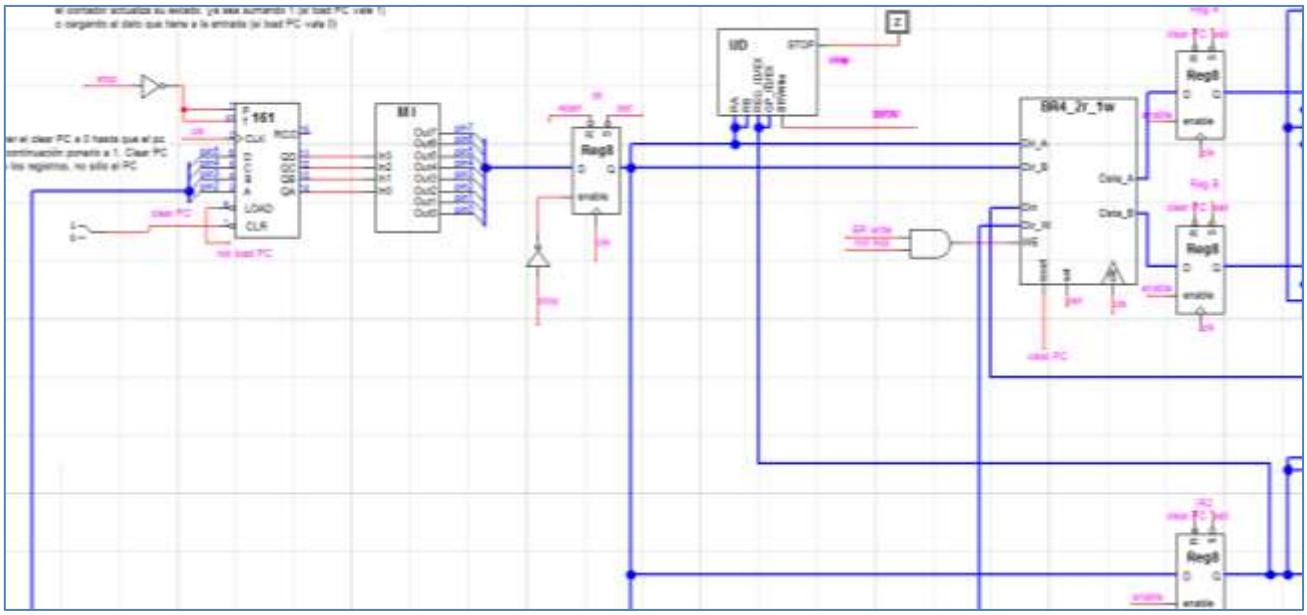
La señal *STOP* indica al procesador que debe detenerse, lo cual significa a nivel hardware:

1. Los valores P y T del PC pasan a valer 0.
2. El valor *Enable* del Reg8 IR pasa a valer 0.
3. El valor del *Mux* que controla las señales de control pasa a valer 0, cargando las siguientes señales:

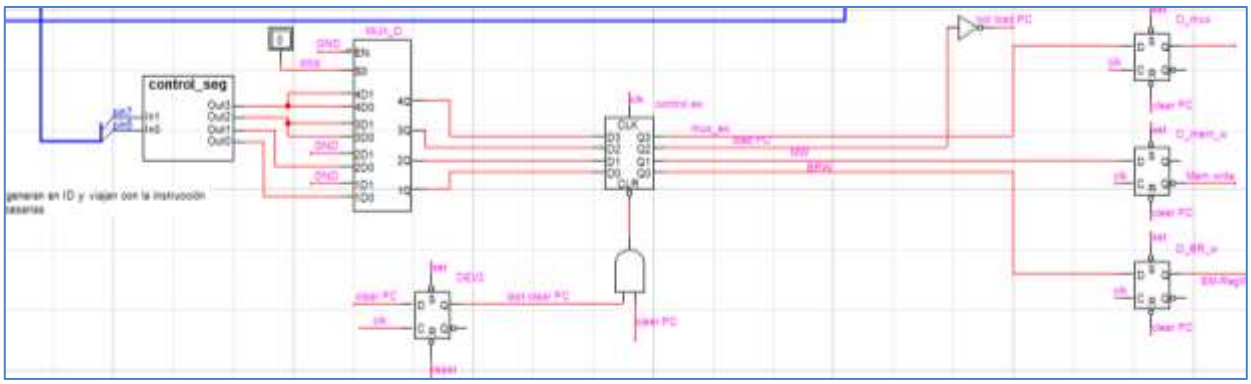
MUX	LPC	MW	BRW
X	X	0	0

   Esto hace que la misma instrucción viajará por el procesador sin escribir ni en memoria ni en el banco de registros.

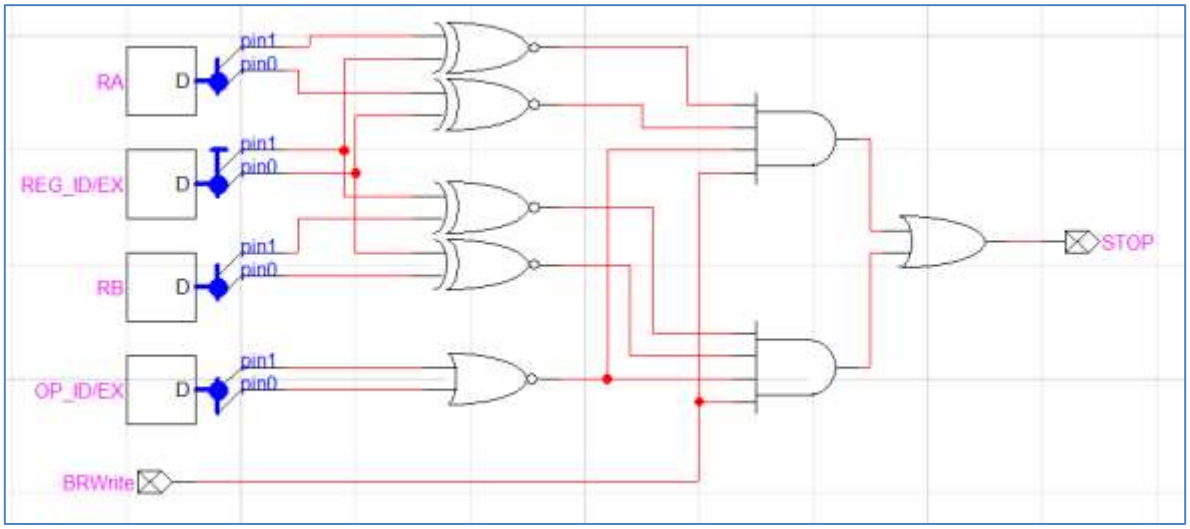
La implementación de la Unidad de Detención ha sido la siguiente:



Procesador



Señal de Control



Unidad de Detención

La ventaja que ofrece la detención del segmentado es la eliminación de riesgos de datos en la operación Load:

Op	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9
LD	IF	ID	EX	MEM	BW				
LD		IF	ID	EX	MEM	BW			
ST		IF	ID	**STOP**	EX	MEM	BW		
JMP		IF	ID	EX	MEM	BW			
ADD		IF	ID	**STOP**	EX	MEM	BW		

Comprobación funcional y temporal

Para comprobar el correcto funcionamiento de las nuevas características del procesador hemos realizado un código que lo compruebe. El código es el siguiente:

Memoria inicial:
[#0] = 01
[#1] = 02

INSTRUCCIONES	CÓDIGO	OPERACIÓN
LD R0,[#0]	00	R0 = 01
LD R1,[#1]	05	R1 = 02
ADD R0,R1	C4	R0 = 01+02=03
ST R0,[#2]	48	[#2] = 03
LD R2,[#2]	0A	R2 = 03
ADD R2,R0	C2	R2 = 03+03 =06
ADD R2,R0	C2	R2 = 06+03 = 09
LD R2,[#2]	0A	R2 = 03
ST R2,[#3]	4E	[#3] =03
NOP	E0	NOP
ADD R0,R0	C0	R0 = 03+03 = 06
JMP [#0]	80	SALTO A 0
NOP	E0	NOP
ST R0,[#4]	50	[#4] = 06

Memoria final:
[#0] = 01
[#1] = 02
[#2] = 03
[#3] = 03
[#4] = 06

Hemos realizado otro código para comprobar el tiempo del reloj, centrándonos en la etapa EX:

Memoria inicial:
[#1] = 01
[#2] = FF

INSTRUCCIONES	CÓDIGO	OPERACIÓN
LD R1,[#1]	05	R1 = 01
LD R2,[#2]	0A	R2 = FF
ADD R1,R2	C9	R1 = 01+FF=00
ADD R1,R2	C9	R1 = 00+FF=FF
ST R1,[#3]	4D	[#3] = FF

Memoria inicial:
[#1] = 01
[#2] = FF
[#3] = FF

Speedup

Mirando el código de la práctica 2 se pueden realizar modificaciones en su orden:

CÓDIGO PASO 3	CÓDIGO CORTOCIRCUITO
LDI R0,[#0] (0)	LDI R0,[#0] (0)
LDI R1,[#1] (4)	LDI R1,[#1] (4)
LDI R2,[#2] (5)	LDI R2,[#2] (5)
NOP	ADD RO,RO (1)
ADD RO,RO (1)	ADD R2,R1 (6)
NOP	ADD RO,RO (2)
ADD R2,R1 (6)	JMP #0 (8)
NOP	STI R2,[#4] (7)
ADD RO,RO (2)	STI R0,[#3] (3)
NOP	X
JMP #0 (8)	X
STI R2,[#4] (7)	X
STI R0,[#3] (3)	X

Si reordenamos así el código, podremos ayudarnos de los cortocircuitos. Sin embargo, la unidad de detención no es usada en esta modificación.

$$\text{Speedup} = \frac{(I*CPI*Tc)práctica2}{(I*CPI*Tc)proyecto1} = \frac{13+4*Tc}{9+4*Tc} = \frac{17*20}{13*26} = 1{,}006$$