

[Título del documento]

[Subtítulo del documento]



[Fecha]

[Nombre de la compañía]

[Dirección de la compañía]

Contenido

[1. Introducción 2](#_Toc71820326)

[2. Banco de registros lectura-escritura en un ciclo 2](#_Toc71820327)

[3. Anticipación de operandos 3](#_Toc71820328)

[3.1 Funcionamiento 3](#_Toc71820329)

[3.1 Implementación el logisim 4](#_Toc71820330)

[3.1.1 Lógica interna 4](#_Toc71820331)

[3.1.2 Unidad de anticipación de operandos vista desde fuera 5](#_Toc71820332)

[4. Unidad de Detección de Riesgos y Detención 7](#_Toc71820333)

[4.1 Funcionamiento 7](#_Toc71820334)

[4.2 Implementación el logisim 8](#_Toc71820335)

[4.2.1 Lógica interna load 9](#_Toc71820336)

[4.2.2 Unidad de detención vista desde fuera (solo load) 10](#_Toc71820337)

[4.2.3 Lógica interna load y beq 10](#_Toc71820338)

[4.2.4 Unidad de detención load y beq vista desde fuera 11](#_Toc71820339)

[5. Nuevo programa ensamblador 12](#_Toc71820340)

[6. Problemas durante la práctica 14](#_Toc71820341)

[6.1 Anticipación de operandos 14](#_Toc71820342)

[6.2 Multiplexor en la etapa EX 15](#_Toc71820343)

[6.3 Unidad de detención 16](#_Toc71820344)



# 1. Introducción

El propósito de esta práctica es crear un procesador segmentado que trate de las mejores formas posibles los riesgos que provoca el uso de estos procesadores (dependencias y riesgos entre instrucciones y con la propia estructura del procesador).

Para ello se implementará un banco de registros de lectura-escritura en un único ciclo, una unidad de anticipación de operandos y una unidad de detención.

# 2. Banco de registros lectura-escritura en un ciclo

El banco de registros del procesador segmentado implementado en la práctica anterior no era capaz de escribir y leer en el mismo ciclo por lo que tenía riesgos estructurales.

Ahora el banco de registros debe trabajar con el flanco de reloj opuesto, con flanco de bajada.

Para implementar esta funcionalidad en logisim debemos cambiar todos los registros del banco de registros de “Rising Edge” a “Falling Edge”.

Gracias a este ligero cambio el programa se ahorraría 2 nops:

* Una nop necesaria para separar la instrucción productora “mov $1, #0” y la instrucción consumidora “ld $4, ($1)”
* Una nop necesaria para separar la instrucción productora “add $1, $1, $0” y la instrucción consumidora “beq $1, $2, FIN”.

La ejecución del programa quedaría de la siguiente forma:

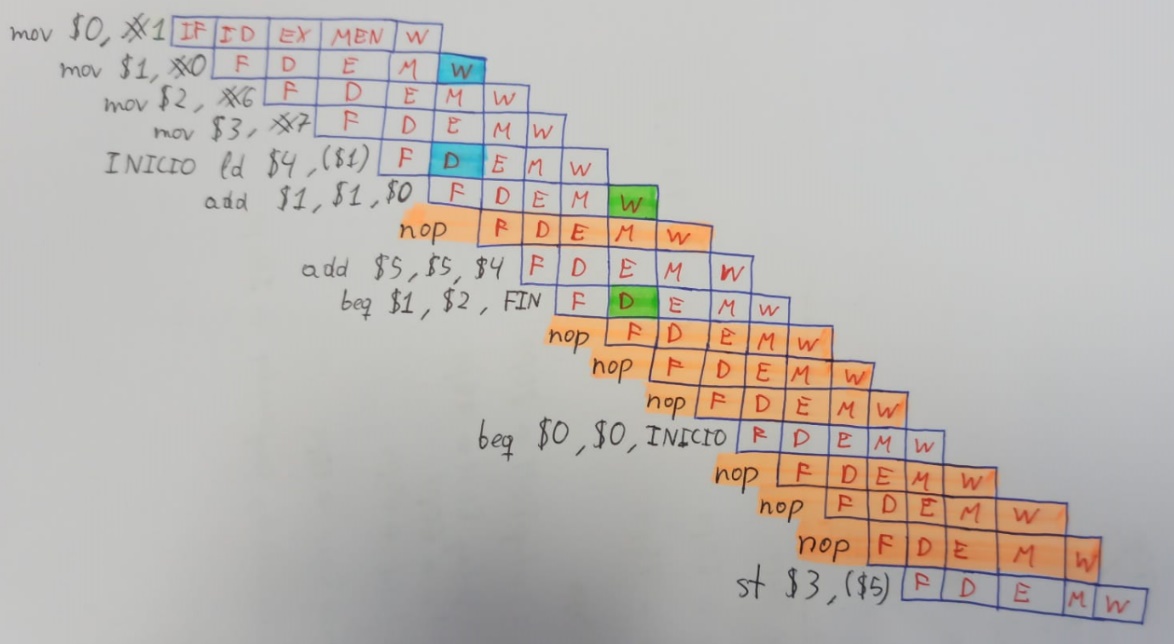


Ilustración 1:Ejecución del programa propuesto utilizando un BR de lectura-escritura en un único ciclo

En azul y verde se ven los dos casos en los que las instrucciones escriben y leen del banco de registros en el mismo ciclo.

En cuanto al código no es posible reorganizarlo de ninguna forma que se ejecute correctamente y nos permita eliminar alguna otra instrucción nop.

# 3. Anticipación de operandos

La anticipación de operandos es una técnica hardware donde se usan los resultados generados por instrucciones antes de que se terminen de ejecutar las mismas. Esto nos ayuda a mejorar las prestaciones ya que tenemos los resultados en etapas anteriores y podremos recortar instrucciones nops.

El circuito con la unidad de control quedaría de la siguiente forma:

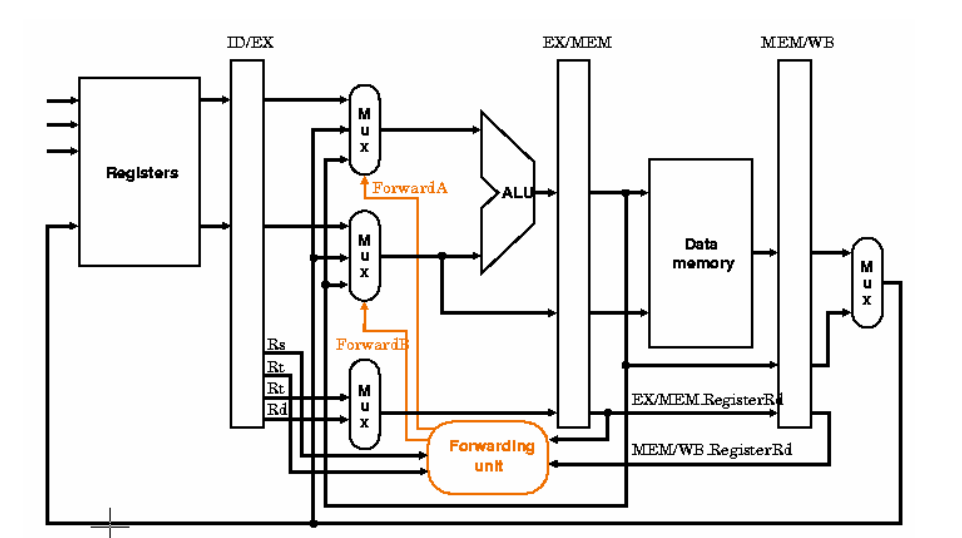


Ilustración 2: Procesador segmentado con una unidad de anticipación de operandos

## 3.1 Funcionamiento

La unidad de anticipación compara cada uno de los operandos del sumador y comparador (ra y rb) con el registro destino de la etapa MEM (EX/MEM.RegistroRw) y con el registro destino de la etapa WB (MEM/WB.RegistroRw).

Si existe igualdad entre un par de ellos y la señal de escritura en registro (RegWr) esta acertada (es 1) en la etapa MEM (EX/MEM.RegWr) o en la etapa WB (MEM/WB.REgWr), la unidad de anticipación debe colocar el valor adecuado para sustituir los operandos de las entradas del sumador o comparador, controlando los multiplexores de anticipación.

Ahora explicaremos su implementación en logisim:

## 3.1 Implementación el logisim

### 3.1.1 Lógica interna

En logisim debemos hacer un circuito independiente y encapsularlo con las siguientes entradas y las siguientes salidas:

Entradas:

* **RegWr(MEN):** Comprueba si la instrucción 1 ciclo anterior escribe en el BR (Se trata de una instrucción de tipo-R, una LOAD o una MOV)
* **RW(MEN):** Obtiene el valor del registro destino de la instrucción 1 ciclo anterior para compararla con Ra y Rb.
* **RegWr(WB):** Comprueba si la instrucción 2 ciclos anterior escribe en el BR (Se trata de una instrucción de tipo-R, una LOAD o una MOV)
* **RW(WB):** Obtiene el valor del registro destino de la instrucción 2 ciclos anterior para compararla con Ra y Rb.
* **Ra** y **Rb**: Sirven para comprobar que registros se están usando en la instrucción actual para comprobar si se tratan alguno de los dos (o los dos) del registro destino de la instrucción previa o la instrucción de dos ciclos atrás. En caso de que haya una igualdad por parte de uno de los dos registros o los dos, se deberá anticipar los datos.

Salidas:

* MuxA y MuxB: Estas señales deciden que datos leer del multiplexor A y del multiplexor B.

La lógica en logisim sería la siguiente:

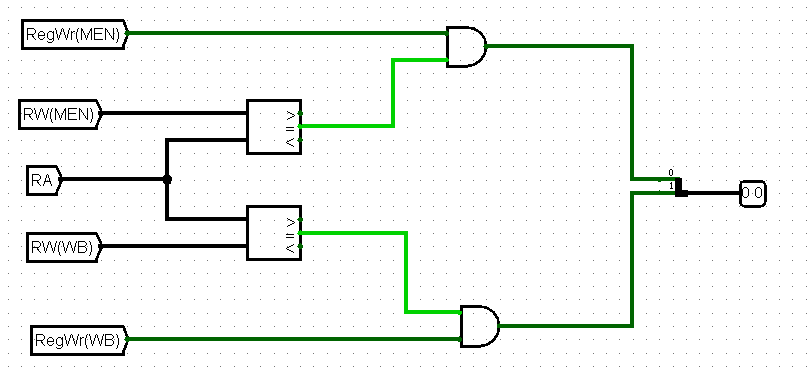


Ilustración 3: Lógica para seleccionar el dato en A

En esta lógica se compara RA con el RW de la etapa MEN y de la etapa WB para generar la salida necesaria que controla el multiplexor de la entrada OP1 del comparador y del sumador.

La salida depende de la comparación de estos registros y la señal de RegWr de la etapa MEN y de la etapa WB.

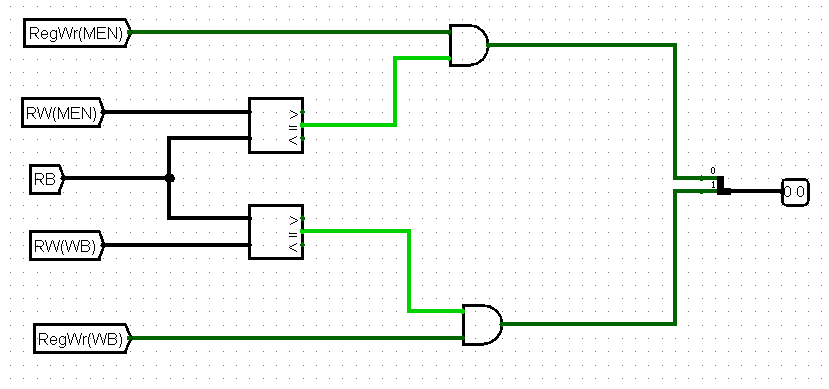


Ilustración 4: Lógica para seleccionar el dato en B

En esta lógica se comprar RB con el RW de la etapa MEN y de la etapa WB para generar la salida necesaria que controla el multiplexor de la entrada OP2 del comparador y del sumador.

La salida depende de la comparación de estos registros y la señal de RegWr de la etapa MEN y de la etapa WB.

3.1.2 Unidad de anticipación de operandos vista desde fuera

El circuito encapsulado lo hemos nombrado “Unidad Anticipación Operandos” y puesto en el procesador segmentado quedaría de la siguiente manera:

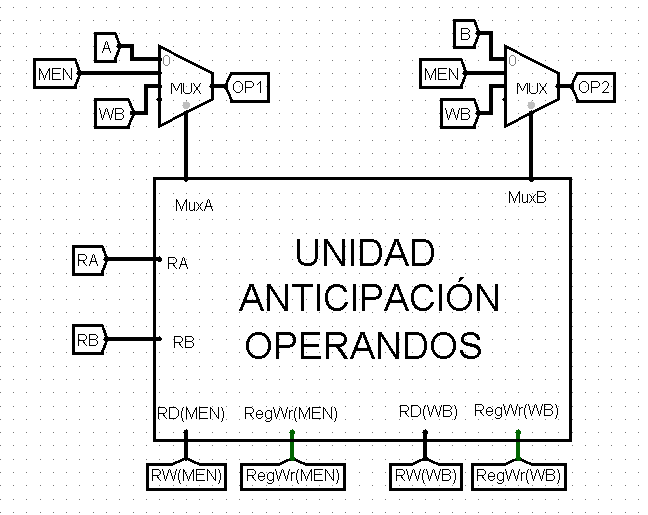


Ilustración 5: Unidad de anticipación de operandos vista desde fuera

Los multiplexores controlan las entradas al sumador en caso de las instrucciones ADD y al comparador en caso de las instrucciones BEQ.

Gracias a la unidad de anticipación de operandos podemos ahorramos una instrucción nop en nuestro programa que había entre la instrucción “add $1, $1, $0” y la instrucción “add $5, $5, $4”. El programa quedaría de la siguiente manera:

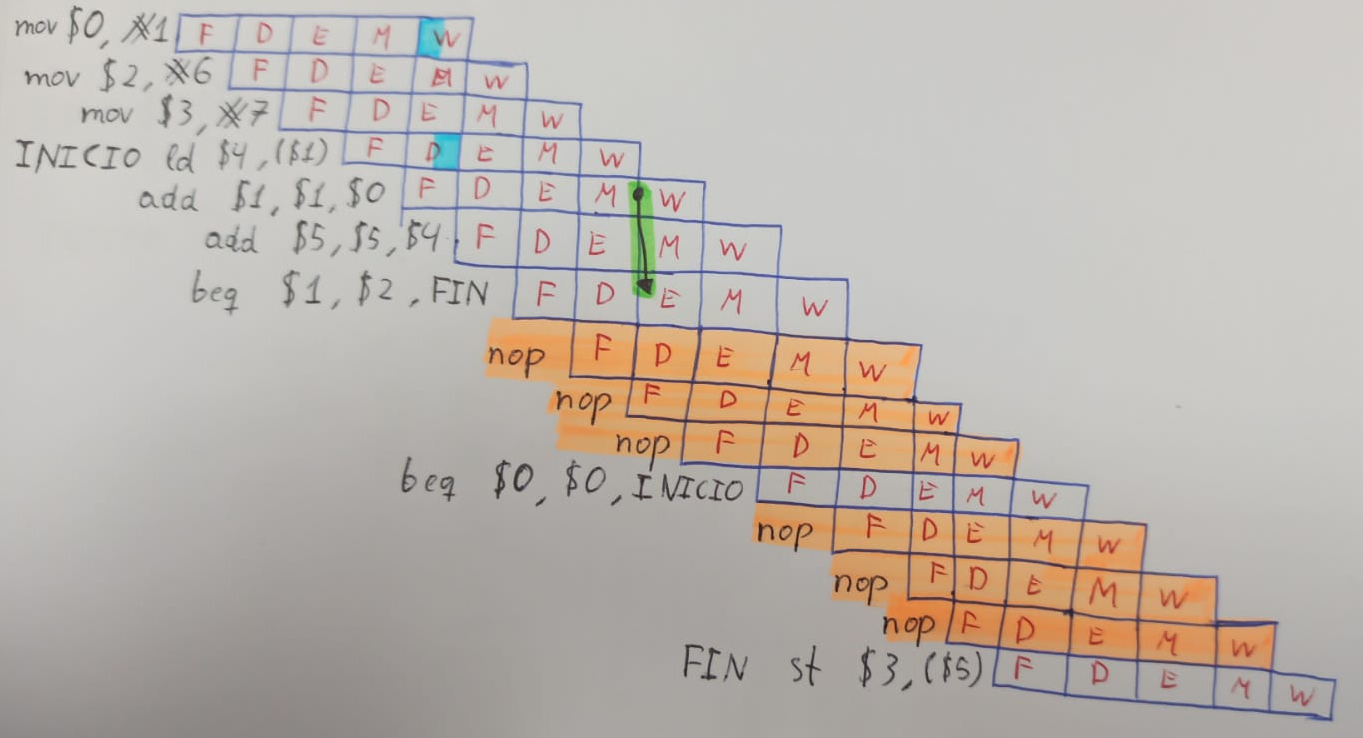


Ilustración : Ejecución del programa con BR lectura-escritura y unidad de anticipación de operandos

En azul se ve como se soluciona el riesgo de leer y escribir en un mismo registro y en verde el riesgo de la anticipación de datos donde se produce un cortocircuito entre la salida de la etapa MEN de la instrucción “add $1, $1, $0” y la entrada del sumador de la entrada

“beq $1, $2, FIN”

# 4. Unidad de Detección de Riesgos y Detención

La anticipación no puede resolver aquellas situaciones en las que se realizará la carga de un registro (LW) y el dato a cargar es un operando de la instrucción siguiente. Como se puede ver en la siguiente instrucción no se puede anticipar los datos:

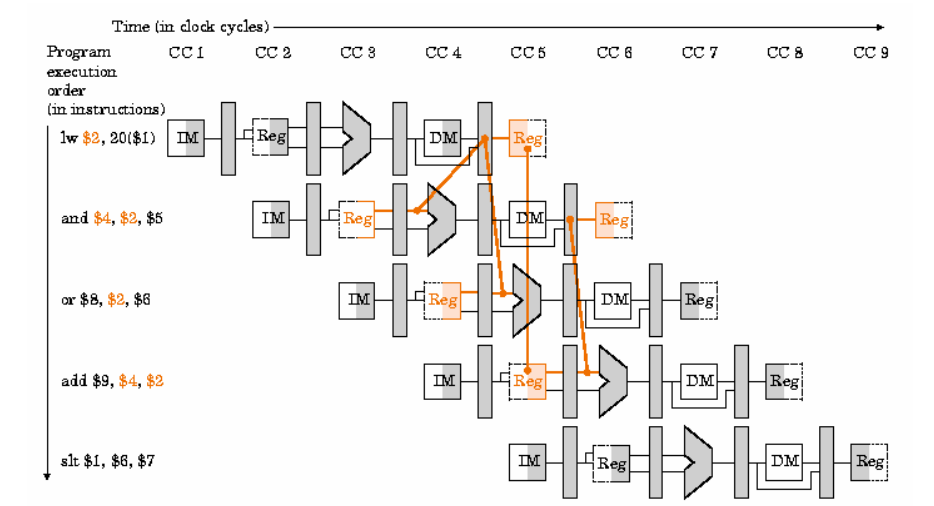


Ilustración 7: Se muestra la ejecución de una secuencia en la que los riesgos no pueden ser resueltos por la unidad de anticipación

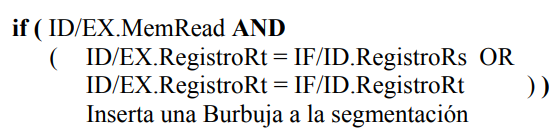
En el ciclo de reloj 4, el dato que se escribirá en $2 se está leyendo de la memoria y la ALU lo requiere en ese mismo ciclo, no es posible anticipar el valor de $2 por que se escribirá en el registro de segmentación hasta el final del ciclo y se podrá disponer de él hasta el siguiente ciclo.

Por lo tanto, se debe detener a la instrucción AND por un ciclo de reloj, para que en el ciclo de reloj 5, con ayuda de la unidad de anticipación, pueda operar sobre el valor correcto de $2.

## 4.1 Funcionamiento

De manera que, además de la unidad de anticipación, es necesario agregar una unidad de detección de riesgos (Hazard detection Unit), que opere en la etapa ID, de manera que pueda insertar una burbuja entra la carga y la instrucción que usará el dato a cargar.

Esta nueva unidad evaluará la siguiente condición:



## 4.2 Implementación el logisim

Para implementar en el procesador segmentado un componente que detecte una carga y justo después una instrucción consumidora de esa carga debemos hacer varias cosas:

* **Cambiar ciertas señales del Contador de programa (PC) y el registro IR:**

Ahora el contador de programa podrá detenerse por lo que las entradas habrá que cambiarlas de la siguiente manera:

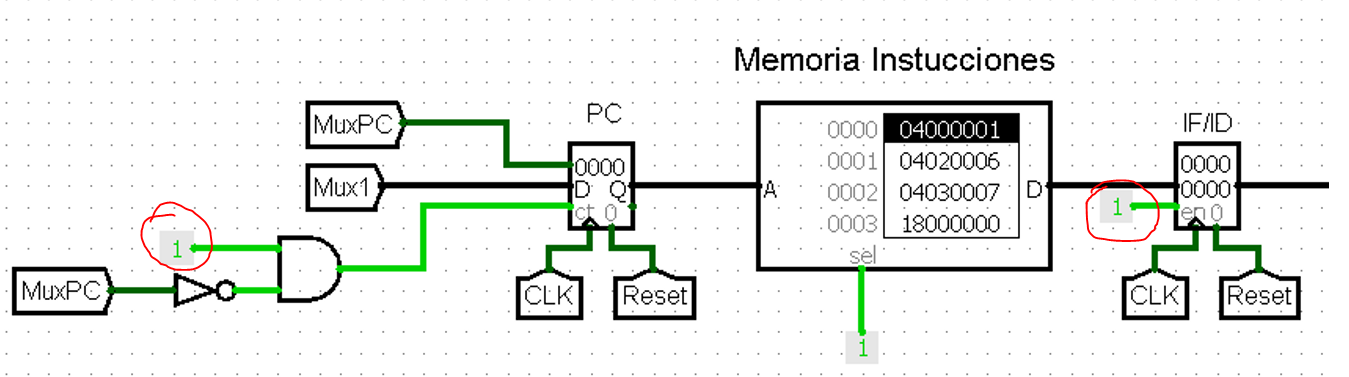


Ilustración 8: Lectura de instrucciones antes

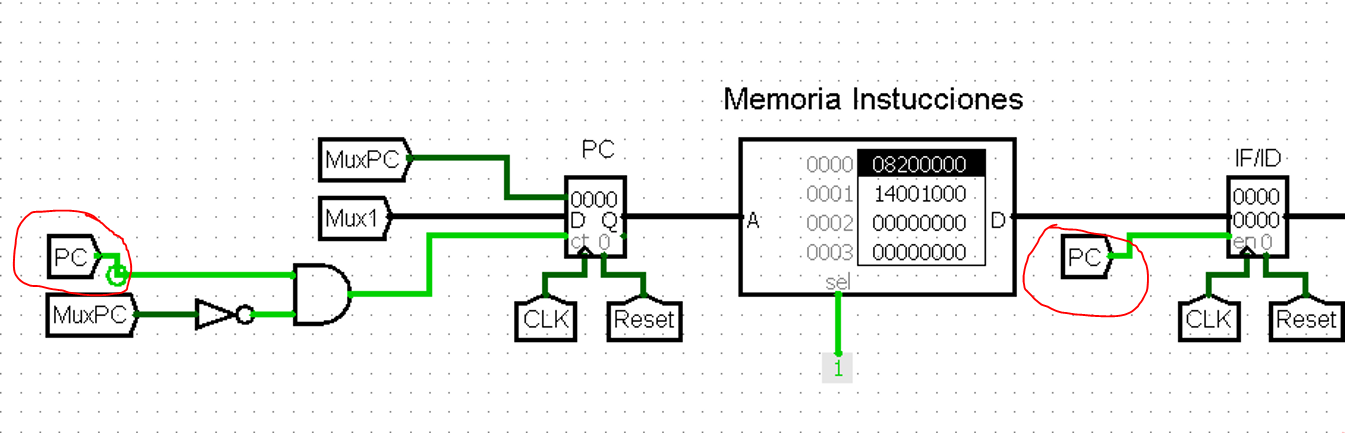


Ilustración 9: Lectura de instrucciones después

Como se puede comprobar una de las entradas de PC que hacía que incrementase antes estaba a “1” siempre (ya que en cada ciclo debía aumentar). Ahora esa entrada la decide la unidad de detención para incrementar solo el contador en caso de que no sea necesaria la detención.

También se ha cambiado la entrada que permite la escritura del registro IR (IF/ID) para, en caso de parar el pipeline, no se escriba la siguiente instrucción y por consiguiente se pierda la instrucción que debe volver a ejecutarse.

* **Añadir un multiplexor a la salida de la unidad de control**

Ahora a la salida de la unidad de control debemos colocar un multiplexor que decida entre escribir los datos de la unidad de control en los registros temporales o todos 0 en caso de querer bloquear el pipeline durante un ciclo. El control de este multiplexor viene dado por la unidad de detención.

Quedaría de la siguiente manera:

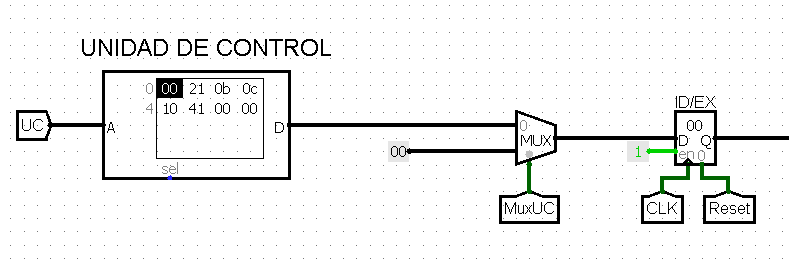


Ilustración 10: Multiplexor que decide que escribir en el registro temporal según la unida de detención.

* **Crear un circuito independiente y encapsularlo.**

### 4.2.1 Lógica interna load

En logisim debemos hacer un circuito independiente y encapsularlo con las siguientes entradas y las siguientes salidas:

Entradas:

* + **IF/ID.RA** y **IF/ID.RB**: Estas señales sirven para comprobar si los registros de la instrucción siguiente son los mismos que el registro destino de la instrucción anterior.
  + **ID/EX.RB**: Esta señal sirve para comparar el registro destino de la instrucción previa con los registros Ra y Rb de la instrucción posterior.
  + **ID/EX.HabilitarMEN**: Esta entrada comprueba si la instrucción ejecutada un ciclo antes accede a la memoria (puede ser una LOAD o una STORE).
  + **ID/EX.RD/WR**: Se debe comprobar que esta señal sea “0” en el caso de que **ID/EX.HabilitarMEN** esté a “1” ya que significaría que se trata de una LOAD y no de una STORE.

Salidas:

* + **MuxPc**: Sirve para aumentar o no el contador de programa.
  + **PC**: Sirve para que se pueda aumentar el contador o escribir en el registro IR.

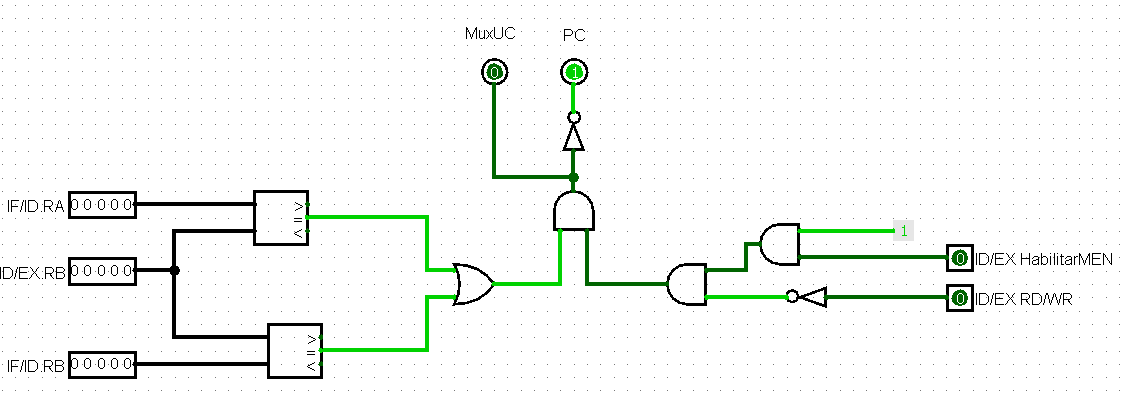


Ilustración 11: Lógica combinacional de la unidad de detención.

### 4.2.2 Unidad de detención vista desde fuera (solo load)

Por fuera el circuito quedaría de la siguiente forma:

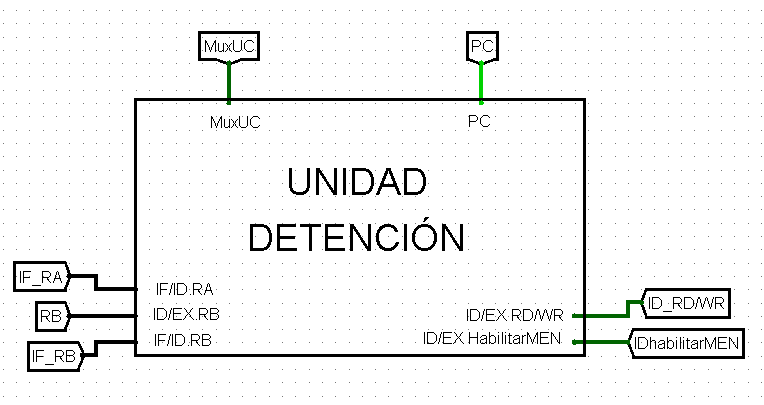


Ilustración 12: Unidad de detención vista desde fuera

### 4.2.3 Lógica interna load y beq

Al acabar la unidad de control nos hemos dado cuenta de que las instrucciones BEQ también deberían ser tratadas por esta unidad de detención para que se bloquee durante 3 ciclos y así no ejecute las instrucciones que haya debajo (nops). Por lo tanto, tenemos que detectar si la señal de control “Beq” está activada en la etapa “EX” y “MEN” para bloquear el pipeline.

Dicho esto, se deberán añadirse estas señales como nuevas entradas de la unidad de detención y tratarlas con el siguiente circuito:

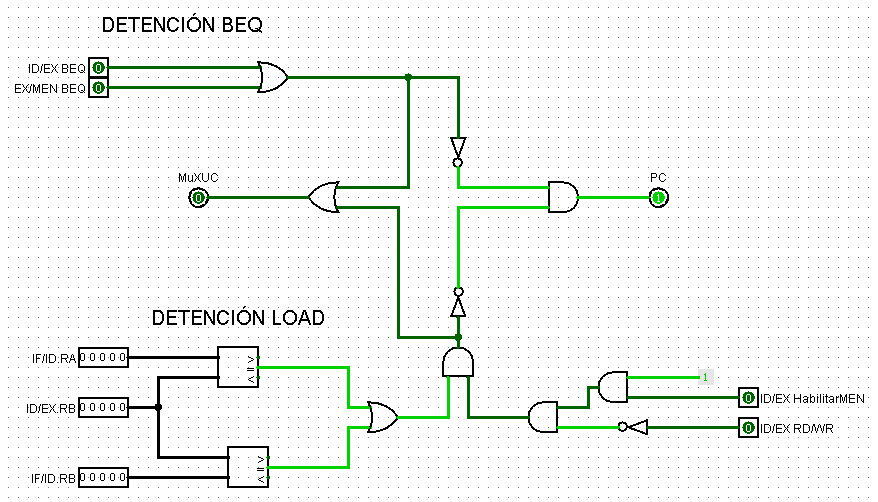


Ilustración : Lógica combinacional de la unidad de detención.

Al circuito mencionado en la ilustración 10 se añaden dos entradas “BEQ en la etapa EX” y “BEQ en la etapa MEN” cuyo funcionamiento es el siguiente:

Si una de las dos entradas es “1” la salida “PC” se pondrá a “0” para que no se pase a la siguiente instrucción y “MuxUC” a “1” para que se escriban ceros en los registros temporales que guardan las señales de control (así se simula una burbuja).

### 4.2.4 Unidad de detención load y beq vista desde fuera

Desde fuera, este circuito se vería de la siguiente forma:

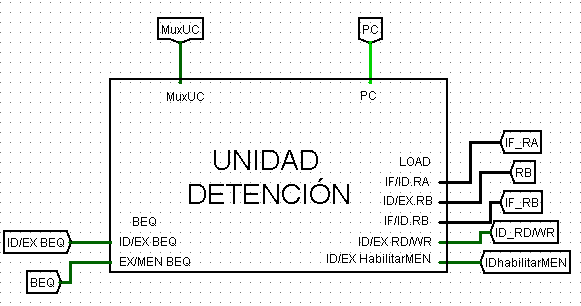


Ilustración : Unidad de detención vista desde fuera

La ventaja principal de este tipo de unidad es que hace la misma función que añadir instrucciones nops con la ventaja de que esta estrategia es a nivel hardware y no a nivel software por lo que el programador puede desentenderse.

Tras implementar esta unidad de detención ya no nos haría falta incluir instrucciones nops para bloquear durante un ciclo el pipeline por lo que la faena del software se pasa al hardware. La ejecución del programa tras la implementación del BR lectura-escritura, la unidad de anticipación de operandos y la unidad de detención es la siguiente:

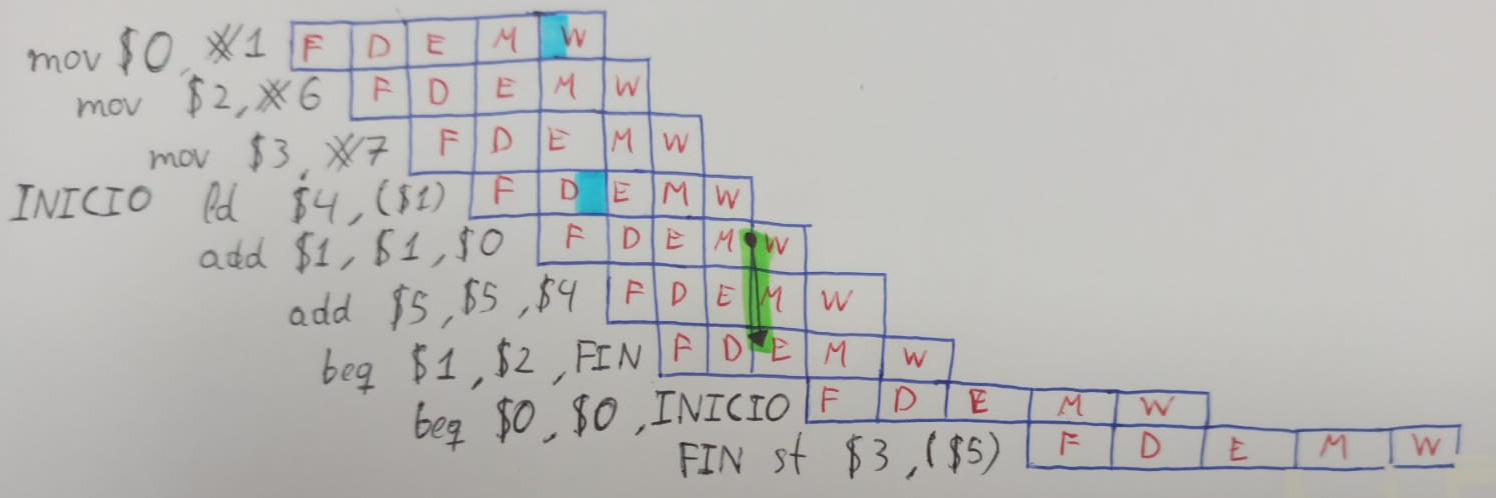


Ilustración : Ejecución del programa con BR lectura-escritura, unidad de anticipación de operandos y unidad de detención

# 5. Nuevo programa ensamblador

En el nuevo programa gracias a la lectura-escritura del banco de registros en un ciclo y a la anticipación de operandos nos ahorramos 3 instrucciones nops :

* 1 Nop entre la instrucción “mov $3, #7” y la instrucción “ld $4, ($1)”
* 2 Instrucciones nops entre la instrucción “add $1, $1, $0” y la instrucción

“add $5, $5, $4”

Antiguamente las instrucciones BEQ necesitaban de la existencia de 3 instrucciones nop para no ejecutar las instrucciones siguientes. Ahora con la implementación de una unidad de control no haría falta a nivel de software poner instrucciones nops ya que de este tema se encargará la unidad de detención.

Por consiguiente, el programa quedaría de la siguiente manera:

mov $0, #1

mov $1, #0

mov $2, #6

mov $3, #7

INICIO

ld $4, ($1)

add $1, $1, $0

add $5, $5, $4

beq $1, $2, FIN

beq $0, $0, INICIO

FIN

St $3, ($5)

En binario quedaría de la siguiente manera:

mov $0, #1

**Binario:**

00000100000000000000000000000001

**Hexadecimal:**

04000001

mov $2, #6

**Binario:**

00000100000000100000000000000110

**Hexadecimal:**

04020006

mov $3, #7

**Binario:**

00000100000000110000000000000111

**Hexadecimal:**

04030007

INICIO

ld $4, ($1)

**Binario:**

00001000001001000000000000000000

**Hexadecimal:**

08240000

add $1, $1, $0

**Binario:**

00010100001000000000100000000000

**Hexadecimal:**

14200800

add $5, $5, $4

**Binario:**

00010100101001000010100000000000

**Hexadecimal:**

14A42800

beq $1, $2, FIN

**Binario:**

00010000001000100000000000001001

**Hexadecimal:**

10220009

beq $0, $0, INICIO

**Binario:**

00010000000000000000000000000100

**Hexadecimal:**

10000004

FIN

St $3, ($5)

**Binario:**

00001100011001010000000000000000

**Hexadecimal:**

0C650000

# 6. Problemas durante la práctica

## 6.1 Anticipación de operandos

Con la nueva anticipación de operandos debemos guardar tanto RA como RB en el registro temporal ID/EX para en la etapa EX compararlos con RW de la etapa MEN o WB

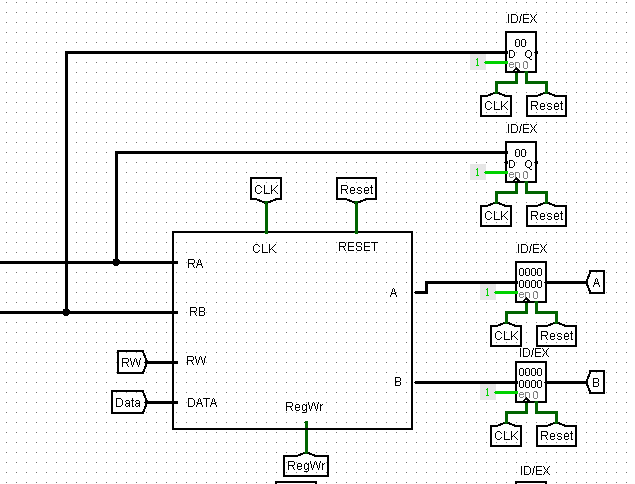


Ilustración 17: Modificaciones en el camino de datos para guardar temporalmente Ra y Rb

## 6.2 Multiplexor en la etapa EX

A la hora de anticipar los datos nos dimos cuenta de que con el diseño del procesador segmentado de la práctica 4 solo podíamos anticipar las instrucciones de tipo R por lo que hemos cambiado ligeramente los componentes añadiendo en la etapa EX un multiplexor que elige que dato se guarda en el registro temporal EX/MEN.

Los datos que se pueden guardar son de los caminos provenientes del sumador (instrucción tipo-R) o del extensor de signo en caso de las instrucciones MOV.

Gracias a este multiplexor podemos cambiar el multiplexor 4:1 de la etapa WB por un multiplexor 2:1.

Para que funcione todo correctamente debemos volver a diseñar las señales de control.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Señales control | | | | | | |
| Código operación | MuxRW | MuxEX | Beq | HabilitarMEN | RD/WR | MuxBR32 | RegWR |
| ADD 101 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| MOV 001 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| LOAD 010 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| STORE 011 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| BEQ 100 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| NOP 110 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

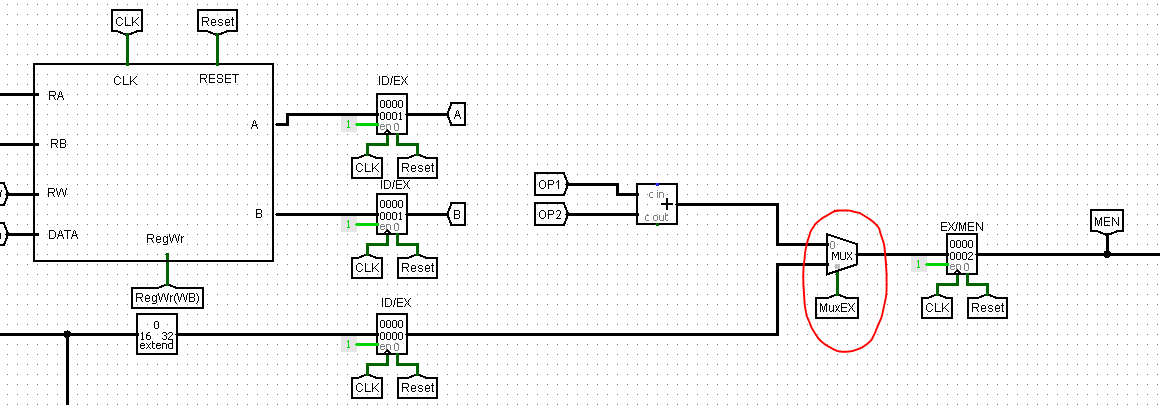


Ilustración 18: Multiplexor añadido en la etapa EX.

## 6.3 Unidad de detención

A la hora de diseñar la unidad de detención que bloquease el pipeline en caso de encontrarse una instrucción BEQ nos dio más de un quebradero de cabeza.

Tuvimos la idea de crear un contador que se iniciara cuando le llegase la señal 100 (señal de la instrucción BEQ) pero nos dimos cuenta de que no era práctico ya que habría que añadir una señal del reloj y una señal reset a la unidad de control y no funcionaba correctamente.