Examen 2 CI-0120

Estudiantes:

- Marco Ferraro **B82957**
- Gabriel Revillat **B86524**

Contenido

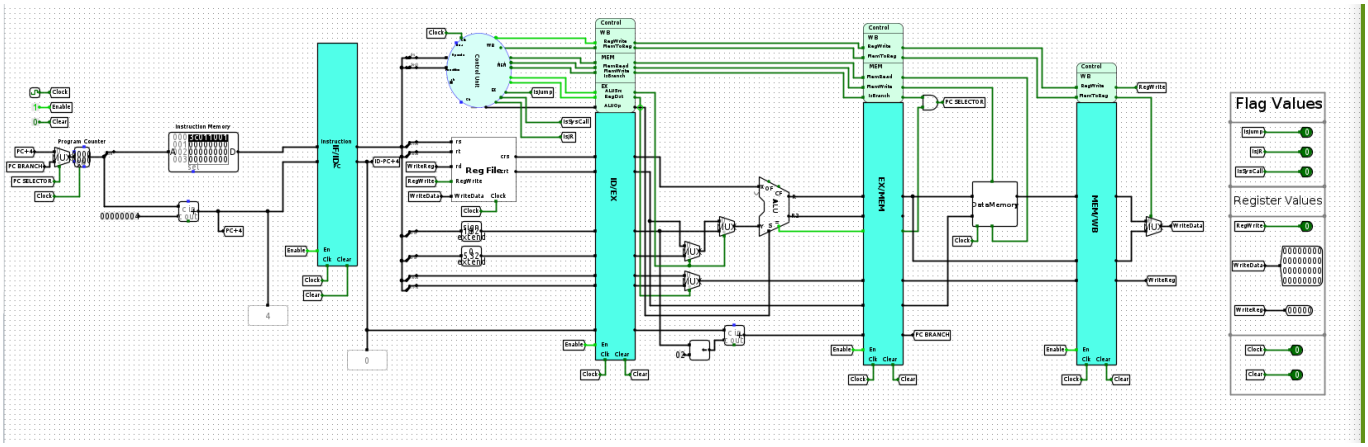
[TOC]

Circuito General

El objetivo de este proyecto es contruir un circuito en **logisim** que simule el funcionamiento de un procesador de arquitectura RISC, además de entender el funcionamiento de *pipelining*.

Logisim es una aplicación desarrollada en Java que nos permite simular el funcionamiento de circuitos lógicos.

Para seguir un buen flujo de trabajo se trabajo con circuitos ya definidos por **logisim**, además de crear un subcircuito para cada componente principal de un CPU RISC.



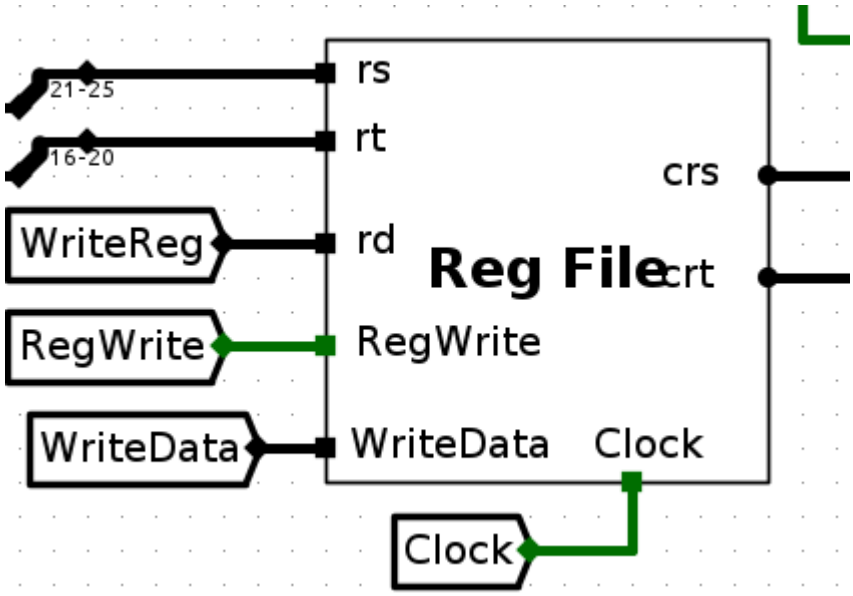
Vista General del Circuito

El circuito maneja un set de instrucciones de **32 bits**, definido en el enunciado del proyecto.

A continuación se mencionarán los principales componentes de del circuito, incluyendo los *stages* de *pipeline* y sus entradas y salidas respectivas.

Register File

El *Register File* es un componente que maneja una matriz de registros dentro del procesador. Además de esto, se puede escribir sobre la memoria de un registro.

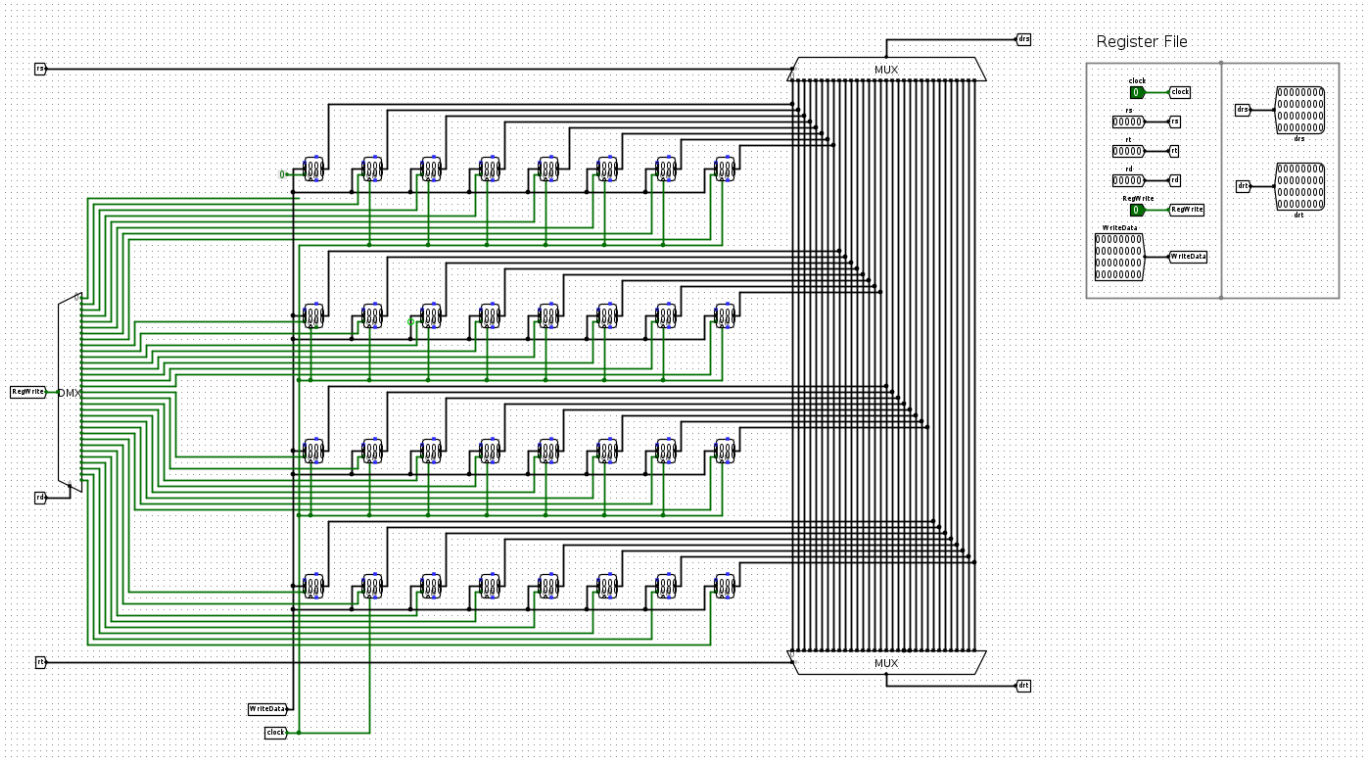


Subcircuito de Register FIle

Cada registro posee **32 bits** de almacenamiento, por lo tanto al *Register File* le ingresan direcciones de 4 bits.

Los registros para visualizar se llaman **rs** y **rt**. Al registro que se le puede modificar la memoria se conoce como **rd**.

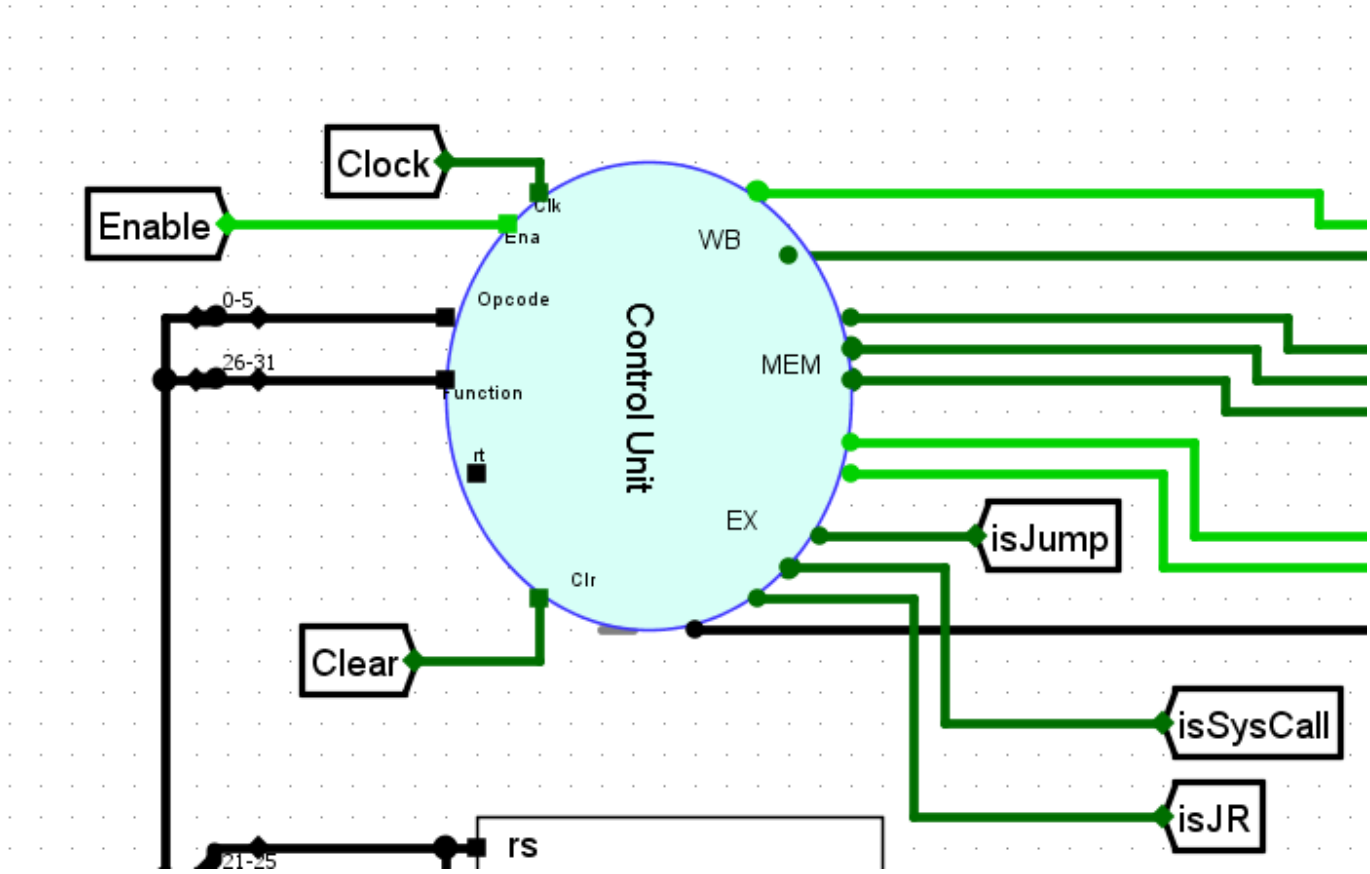
Sus salidas son los valores de los registros **rs** y **rt**



Lógica Interna del Register File

Control Unit

La Unidad de Control en un componente que nos ayuda con la ejecución de las instrucciones. Si manejamos un flujo **Multi Cycle**, la unidad de control deberá manejar varios **Datapaths** y **flags**.



Subcircuito Unidad de Control

Para determinar los valores que cada stage debería de recibir, nos basamos en el formato **R** de intrucciones para procesadores RISC.

	EX				MEM			WB	
Instr.	**RegDst**	**ALUOp 1**	**ALUOp2**	**ALUSrc**	**Branch**	**MemRead**	**MemWrite**	**RegWrite**	**MemToReg**
R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Splitter de Instrucciones

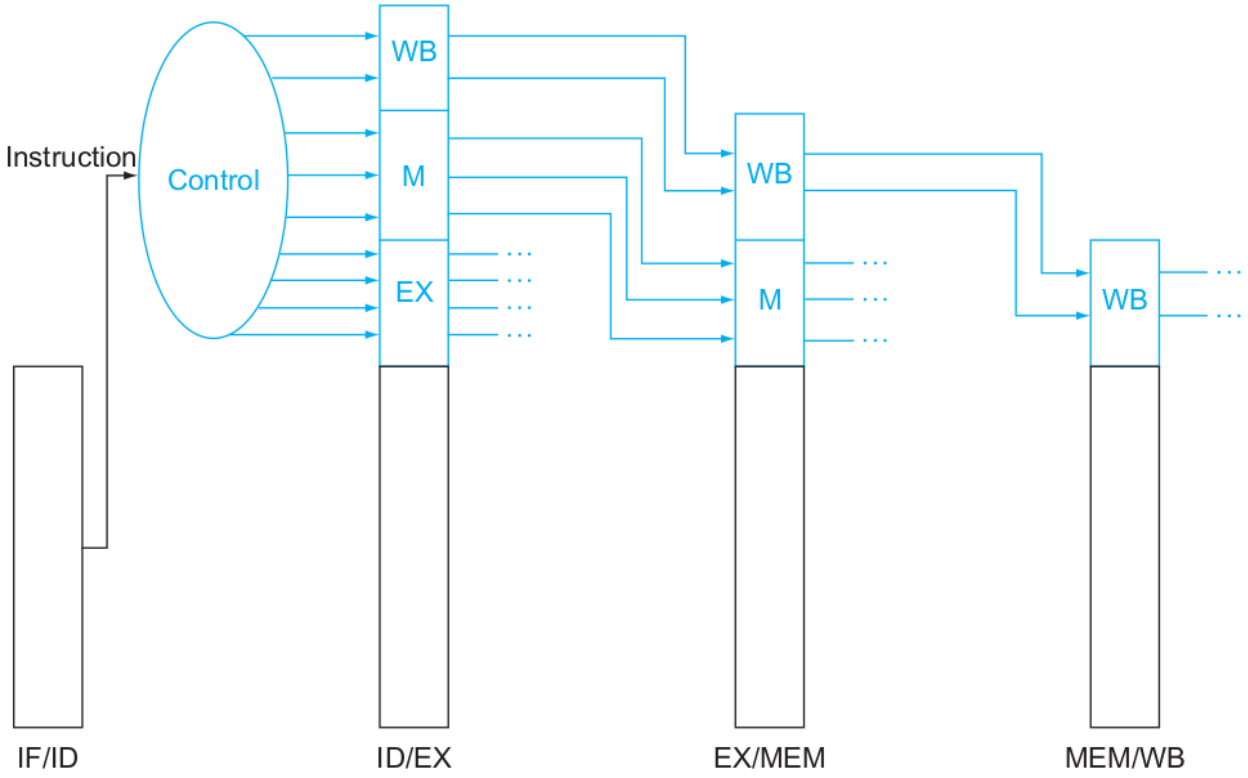
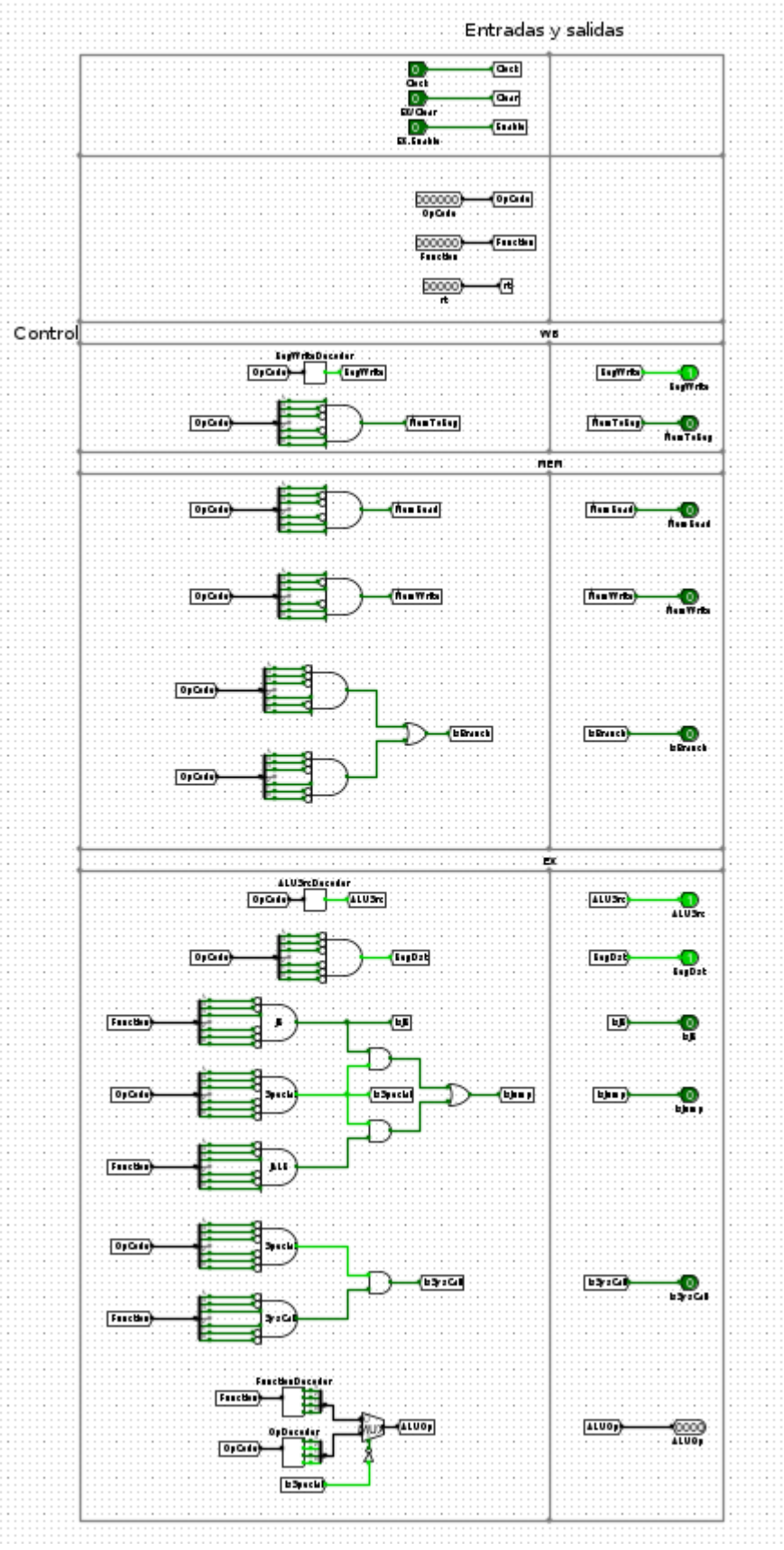


FIGURE 4.50 The control lines for the final three stages. Note that four of the nine control lines are used in the EX phase, with the remaining five control lines passed on to the EX/MEM pipeline register extended to hold the control lines; three are used during the MEM stage, and the last two are passed to MEM/WB for use in the WB stage.

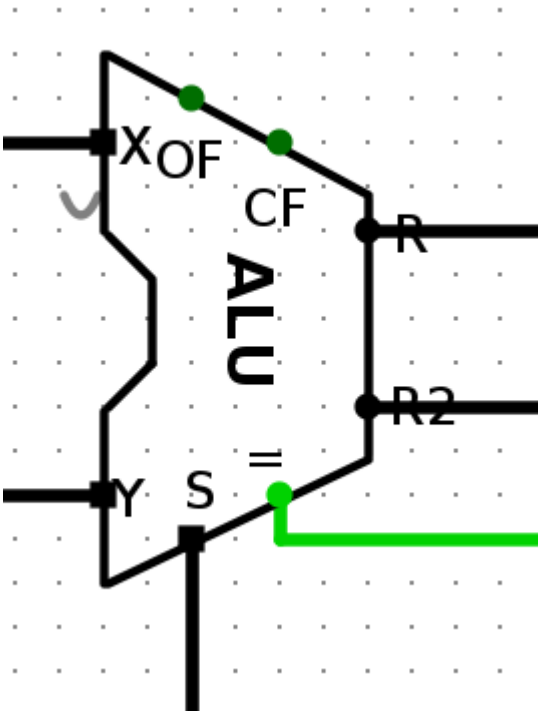
Flujo de los valores de salida para cada Stage



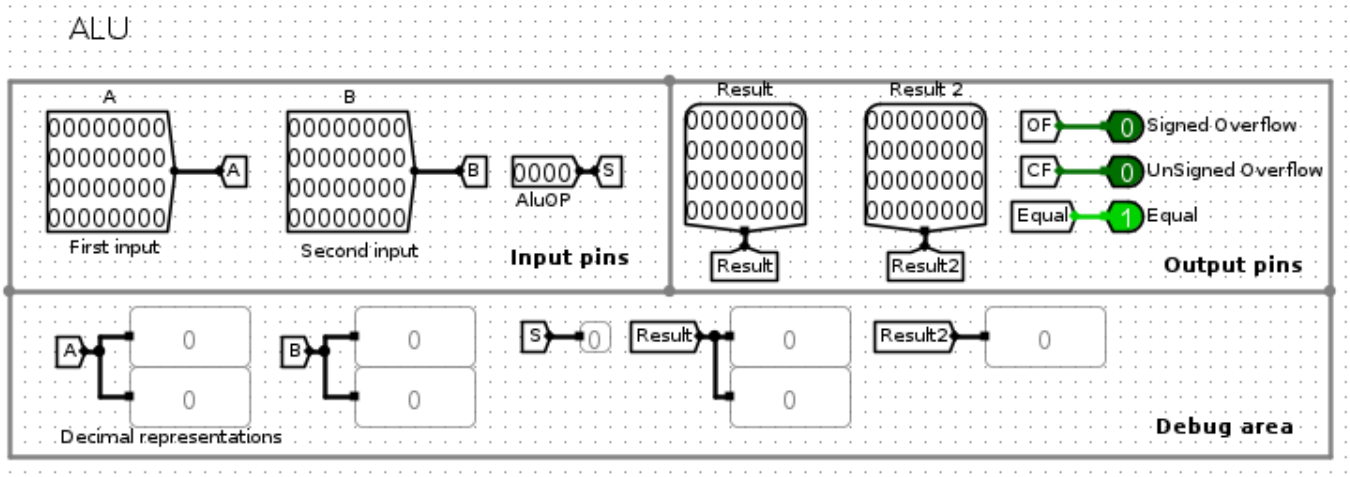
Circuito interno de la Unidad de Control

ALU

La unidad de control es la encargada de expresar operaciones aritméticas y lógicas sobre nuestro CPU. Nuestra UAL o ALU recibe 2 instrucciones de **32 bits**.



Subcircuito ALU



Salidas ALU

Nuestro ALU es de instrucciones reducidas. Notese que el espacio de **AluOP** tiene 4 bits. Las instrucciones básicas que hace es

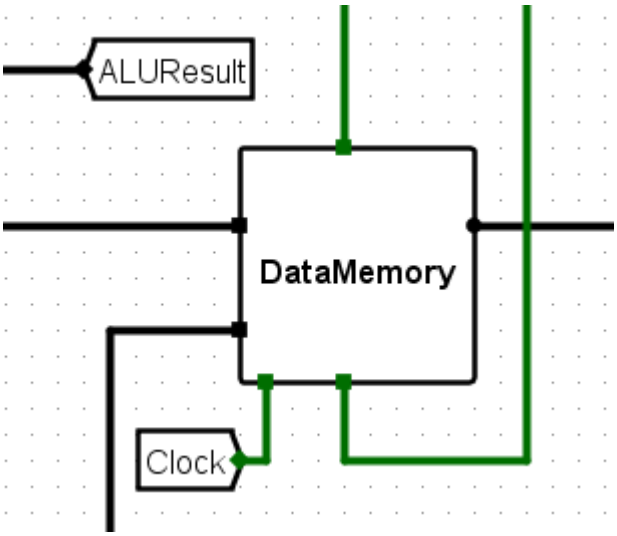
- Suma
- Resta

en la parte aritmética.

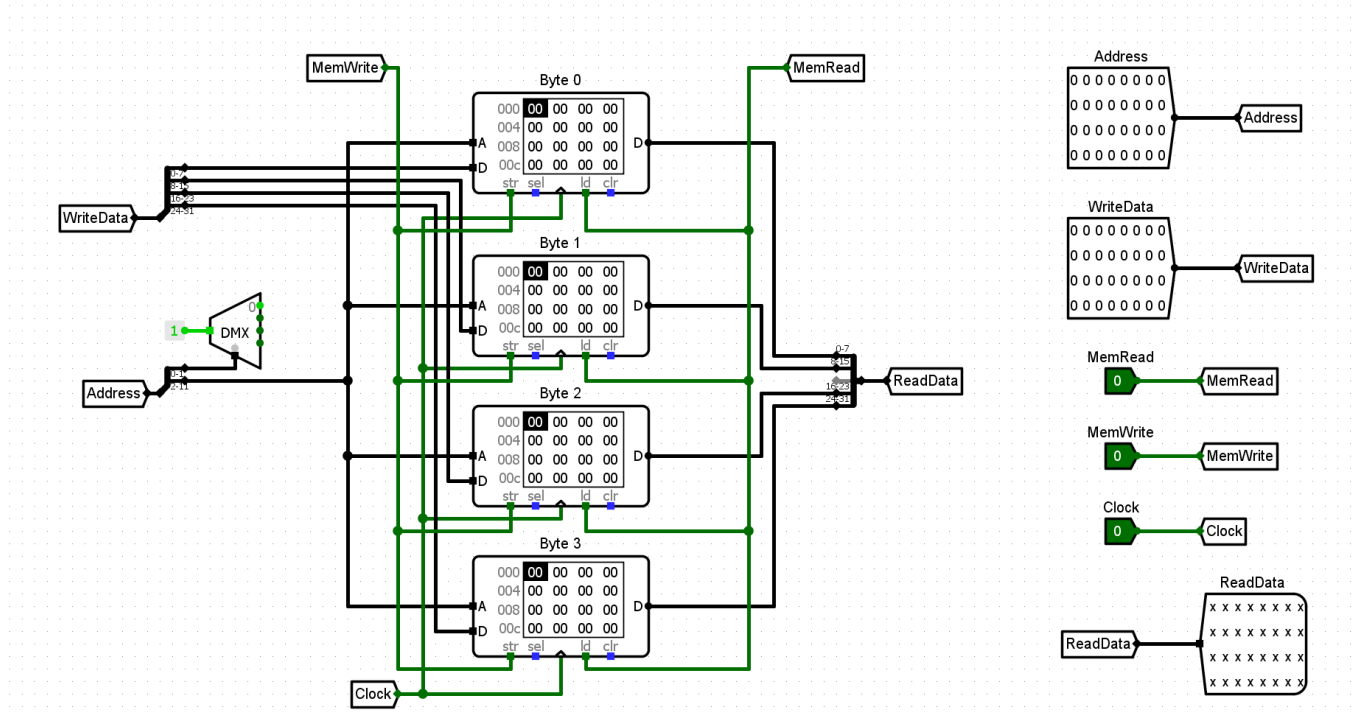
En la parte lógica tenemos una serie de **flags** las cuáles nos sirven para identificar si 2 valores son iguales, si hay un overflow de negativos o positivos.

Data Memory

El componente de data memory es escencial para el fucionamiento del ALU. Si hay una instruccion **load**, existe un multiplexor afuera que va a girar la salida de la intruccion al **DataMemory.** Recordemos que por el **Pipelining**, la memoria a leer viene definida desde la salida del stage anterior.



Subcircuito DataMemory



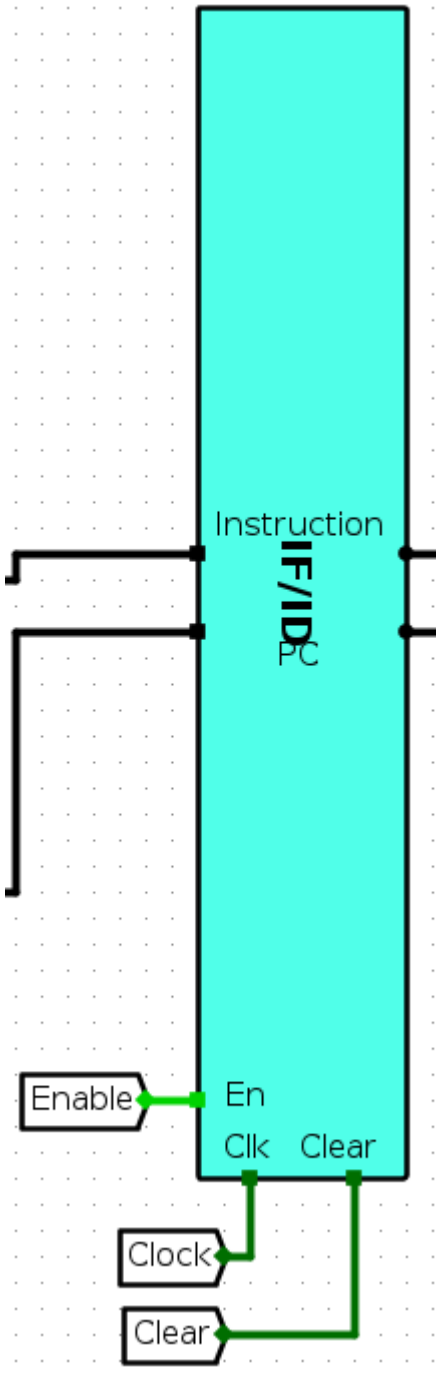
Circuito Data Memory

Nótese que el tunnel llamado **WriteData** es el mismo que se utiliza para el **Register File**

IF/ID

El stage **IF/ID** se encarga del

- Instruction Fetch
- Instruction Decode



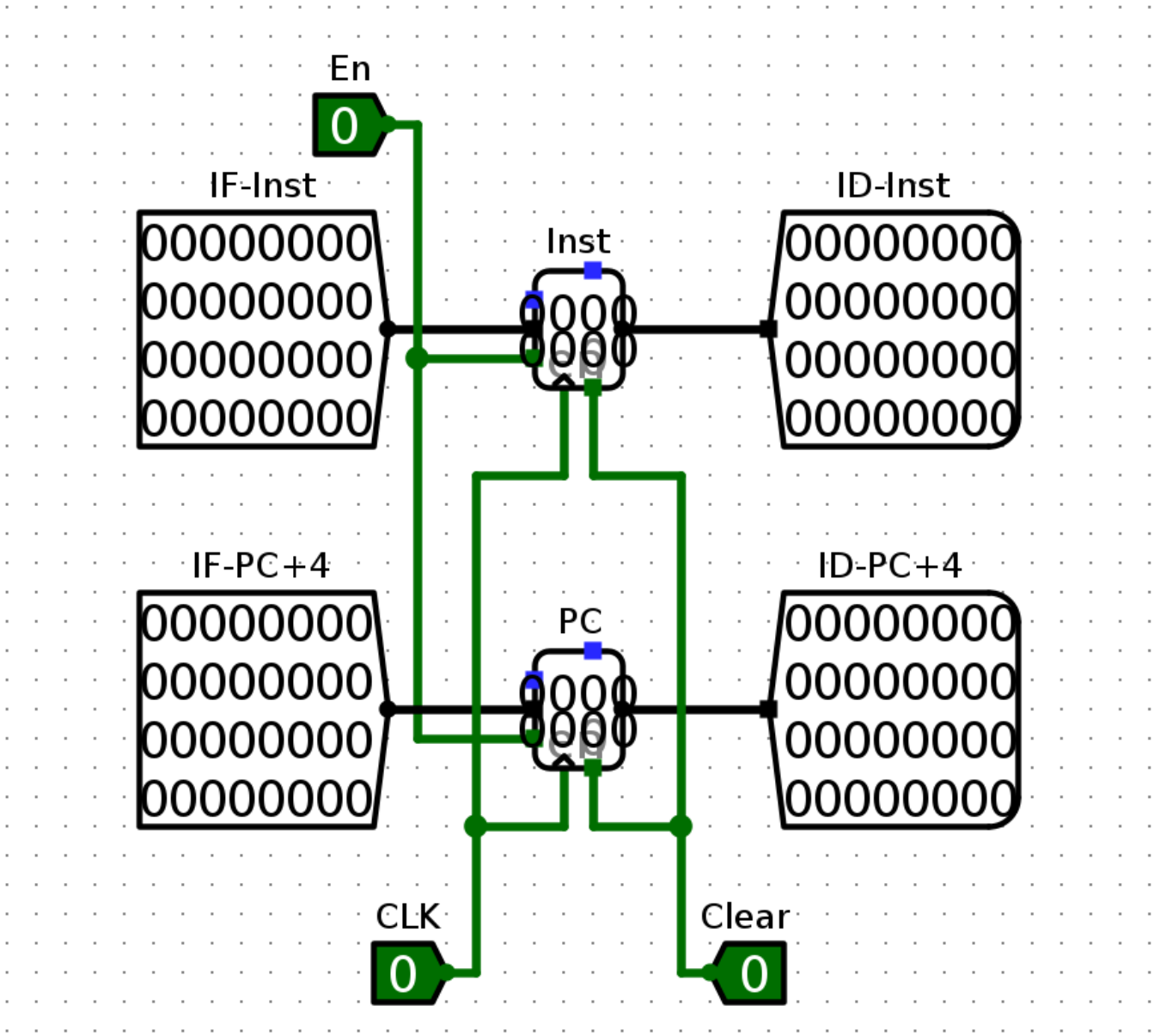
Subcircuito IF/ID

El proceso de **Instruction Fetch** se encarga de:

- Obtener la instruccion desde **Instruction Memory**
- Aumentar el valor de **Program Counter**

Mientras que el proceso de **Intruction Decode**:

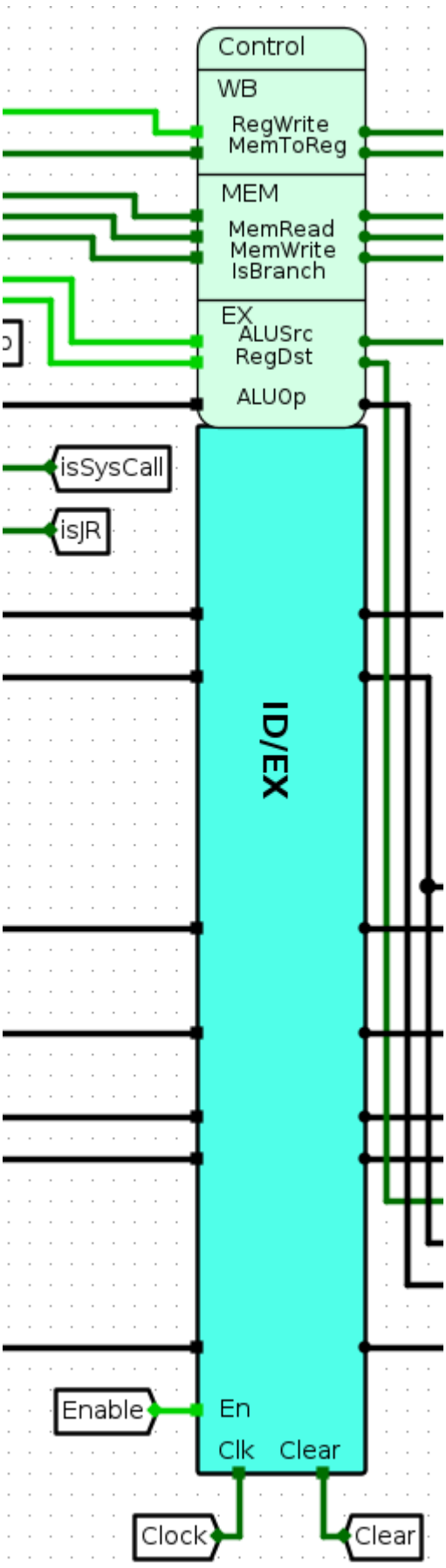
- Obtiene los valores de los registros de la instrucción



Circuito IF/ID

ID/EX

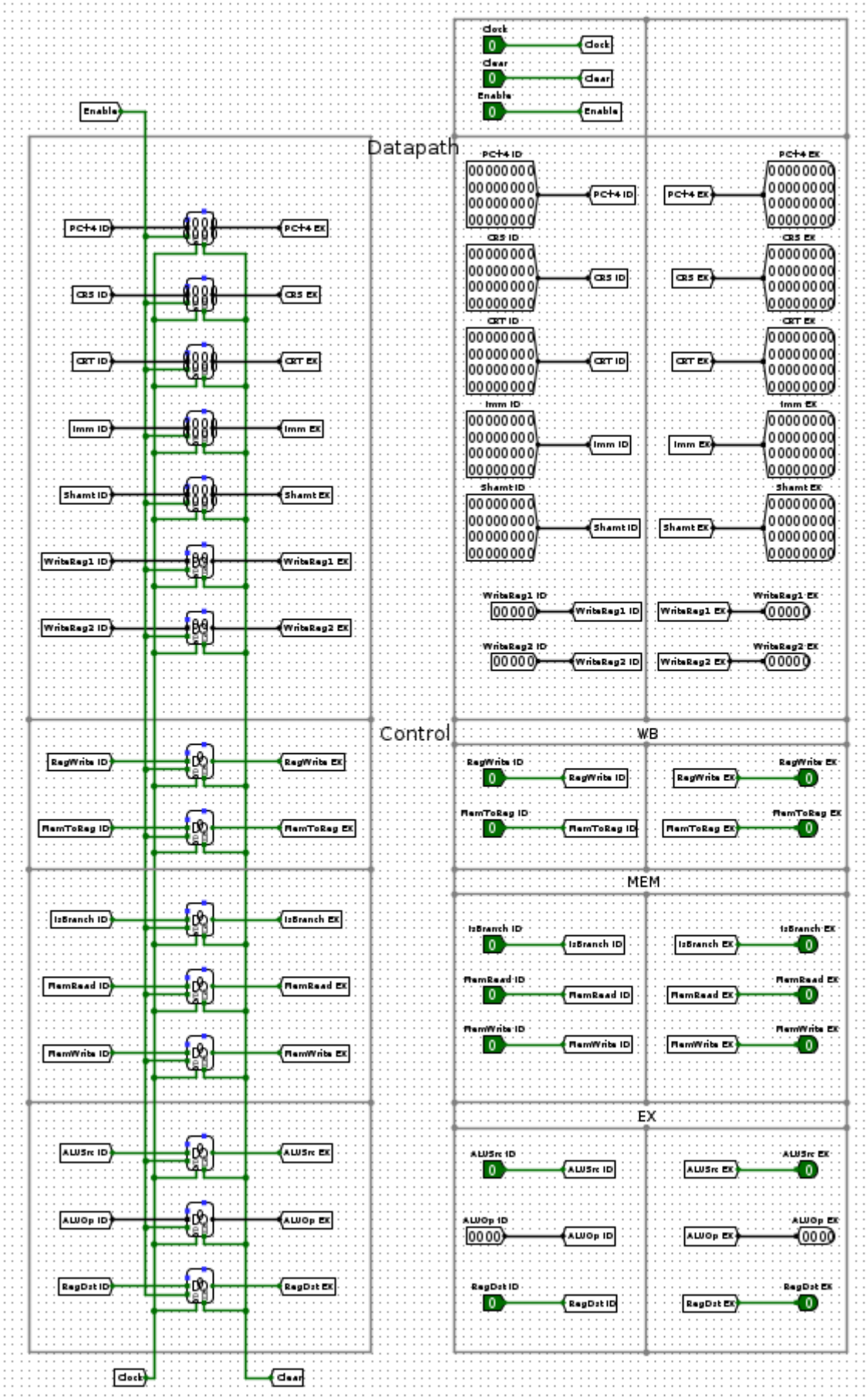
Este stage es más complejo. Ya se explico que hace un proceso **Instruction Decode** anteriormente, por lo tanto solo se va a mencionar el **Execute**.



Subcircuito ID/EX

Execute es un proceso que:

- En caso de ser una referencia de memoria, setea el OFFSET y la Base
- Es caso de recibir una referencia de un operacion aritemetica, hace la mate, por decirlo así.

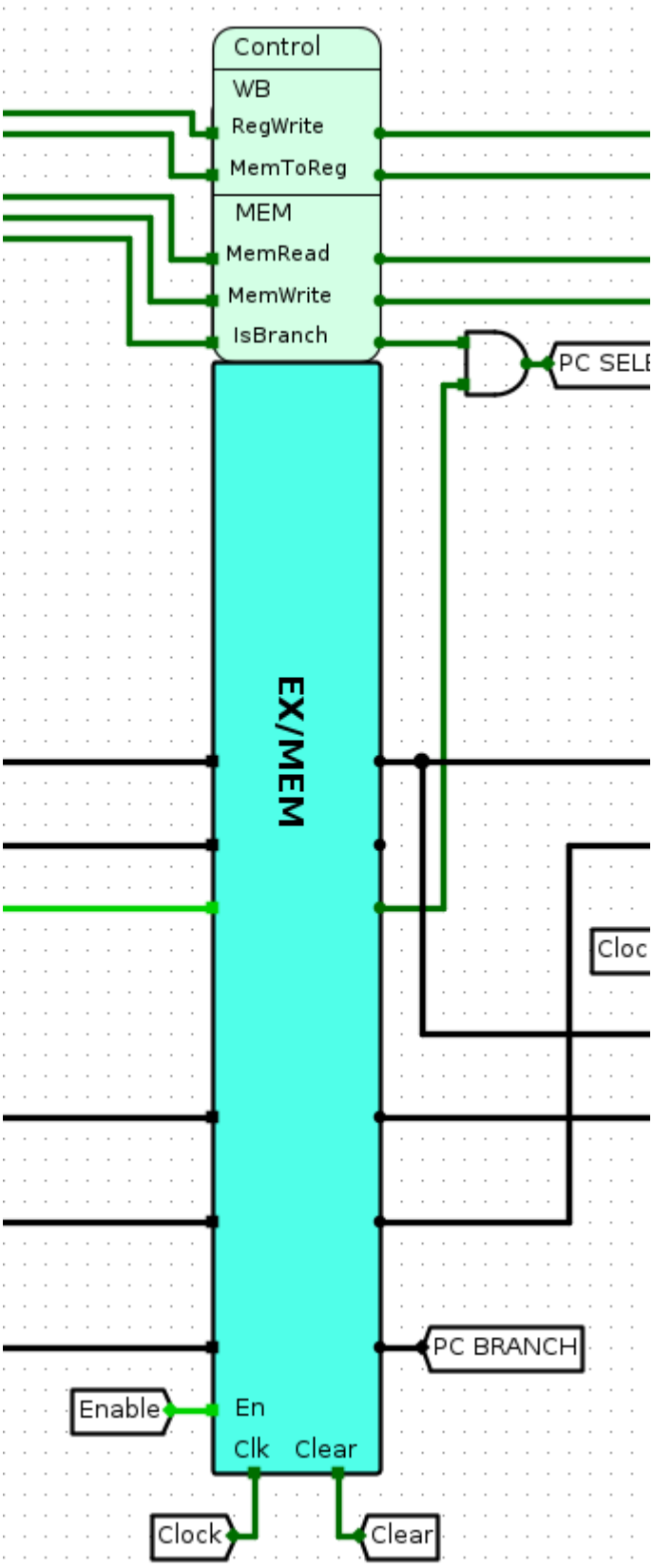


Circuito

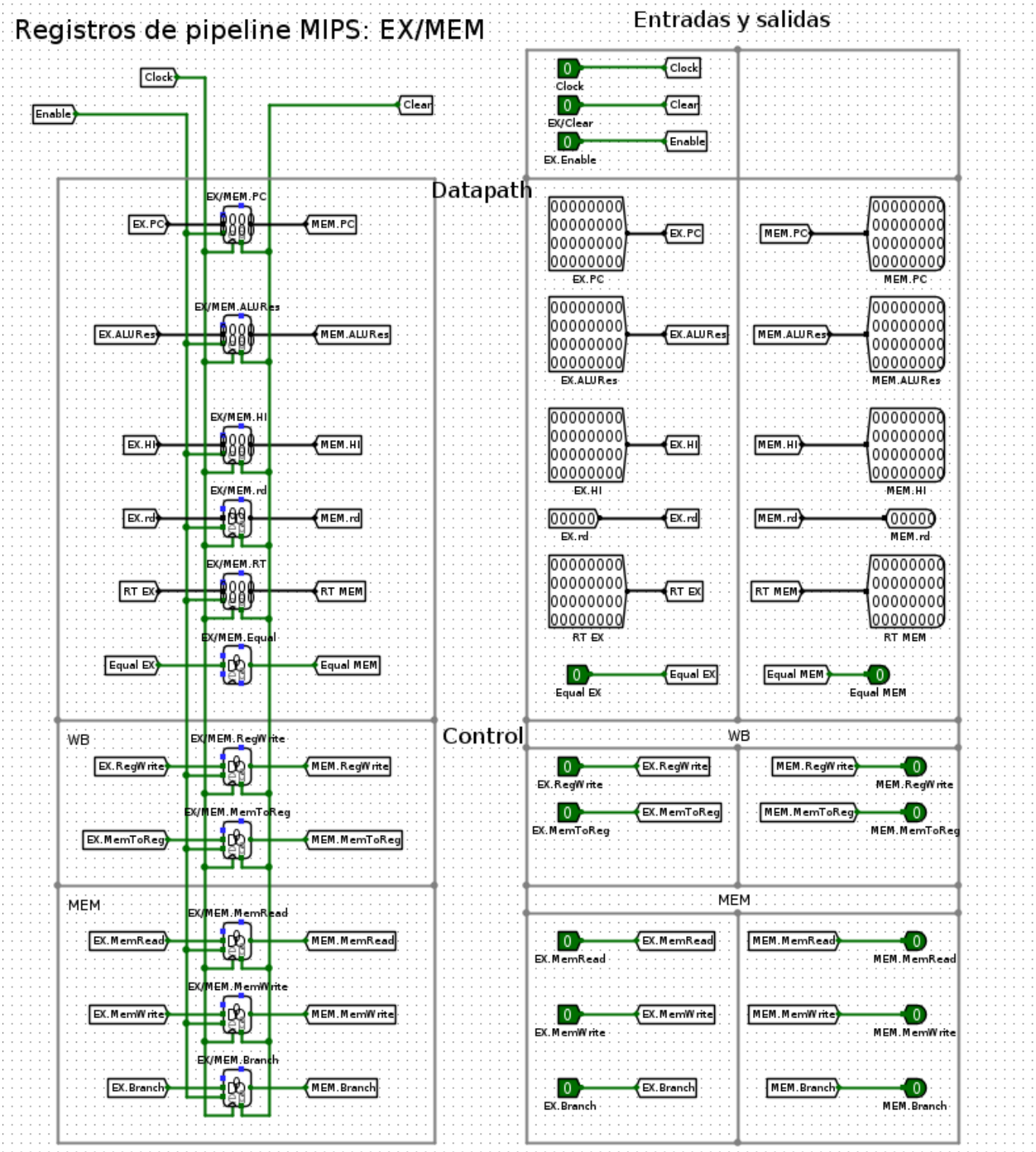
EX/MEM

El proceso **Memory** se encarga de:

- Acceder a memoria
- Si recibe un Branch, modificar el Program Counter



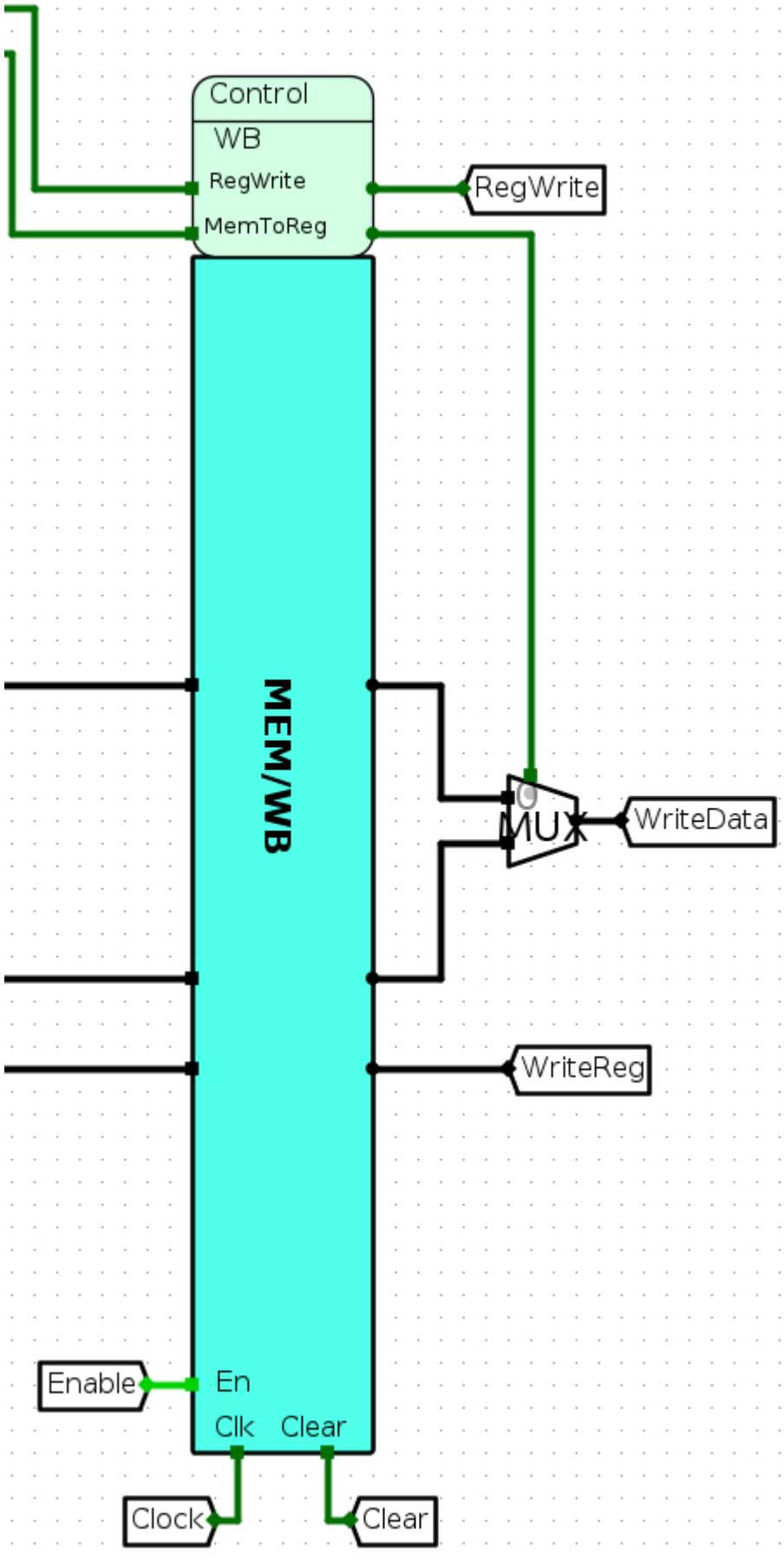
Surcircuito EX/MEM



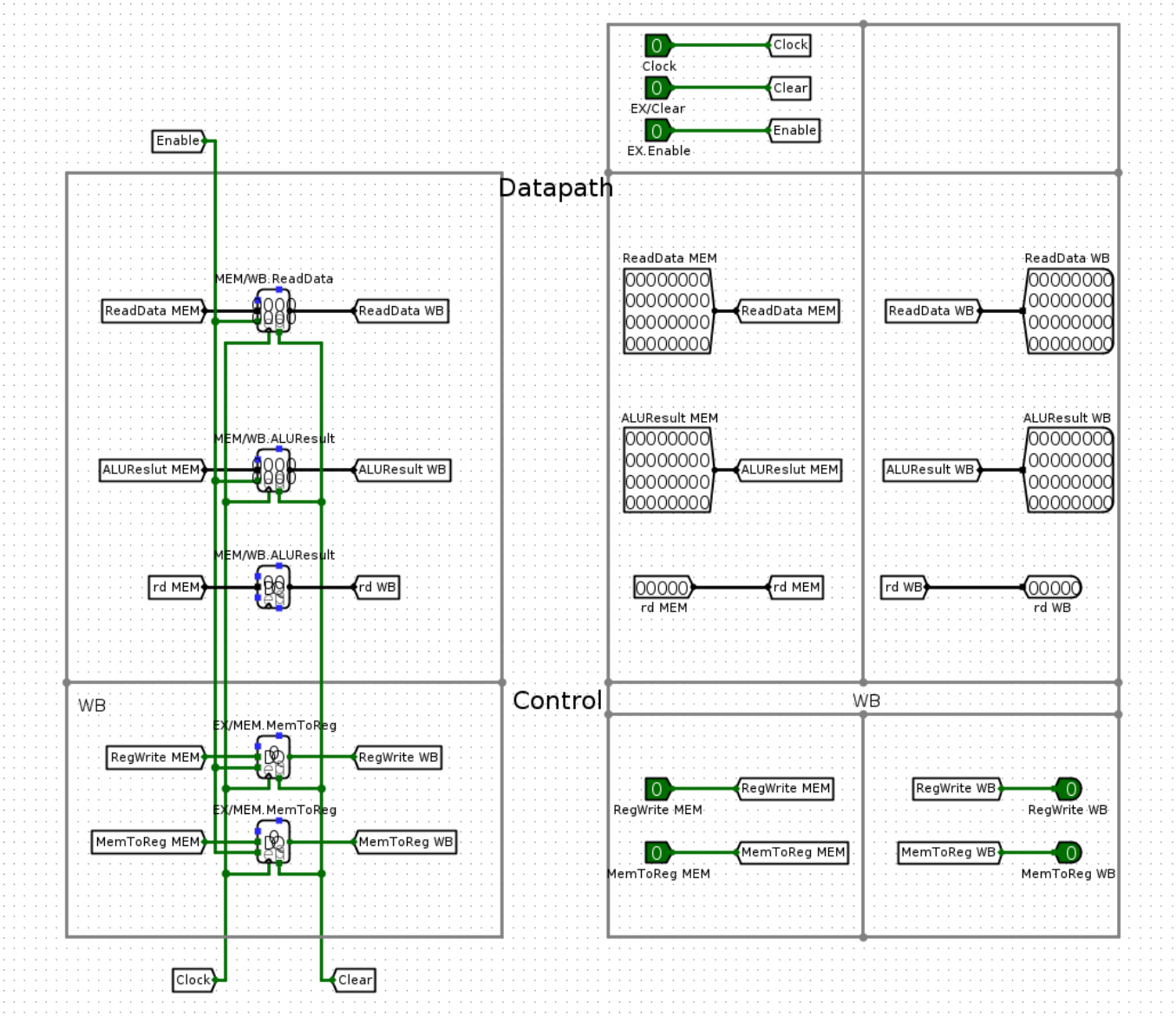
Circuito

MEM/WB

Finalmente, tenemos **WriteBack.** WB se encarga de poner un valor en un registro respectivo, por lo tanto está en comunicación directa con el **Register File.**

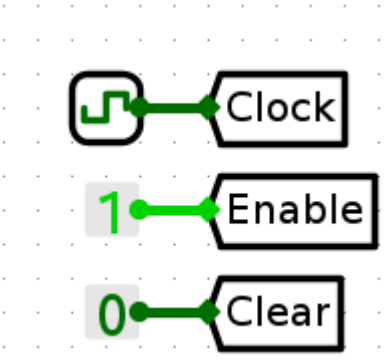


SubCircuito MEM/WB

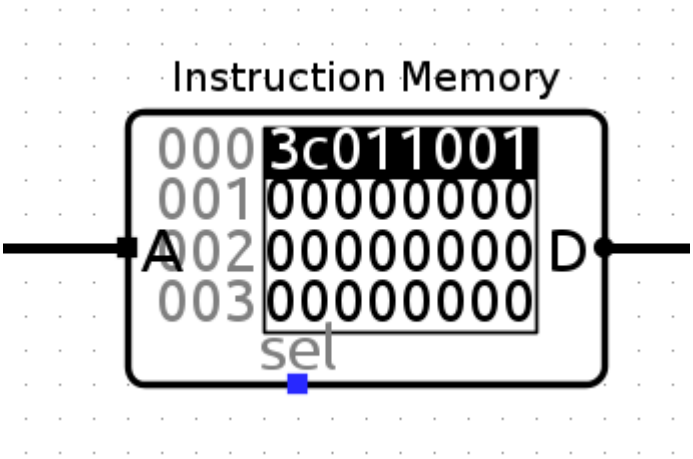


Circuito

Componentes Externos



Reloj global para todo el Circuito



Almacenamiento de las Intrucciones

Pregunta A

Para este punto se tuvo que implementar un **Forwarding Unit**. Esta unidad tiene como funcionalidad enviar señales mediante 2 caminos al **Register File**.

Forwarding Unit

El propósito de la unidad de *forwarding* es garantizar que la instrucción que ingresa a la stage **EX** del *pipeline* reciba los valores correctos para sus registros operandos.

Entradas:

- **ID/EX.rs** y **ID/EX.rt**: 5 bits.
- **EX/MEM.rd** y **MEM/WB.rd**: 5 bits.
- **EX/MEM.RegWrite** y **MEM/WB.RegWrite**: 1 bit.

Salidas:

- **Forward A**: 2 bits. Controla la selección del operando X del ALU.
- **Forward B**: 2 bits. Controla la selección del operando Y del ALU.

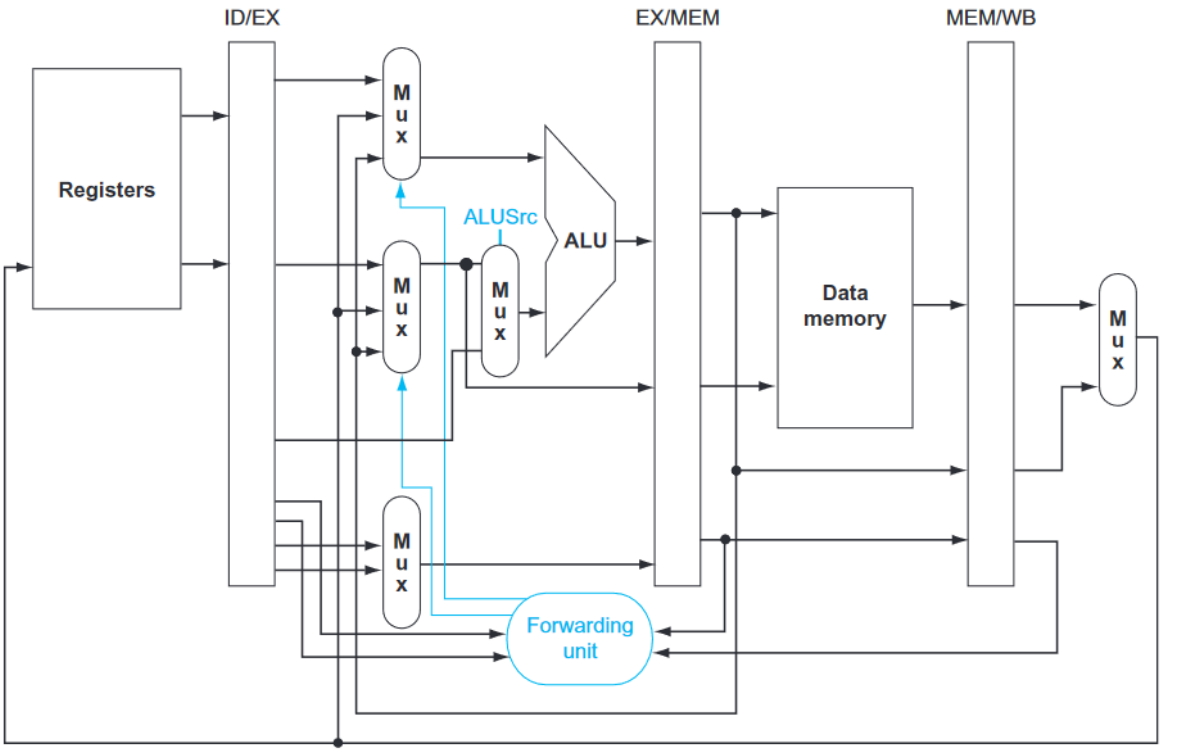
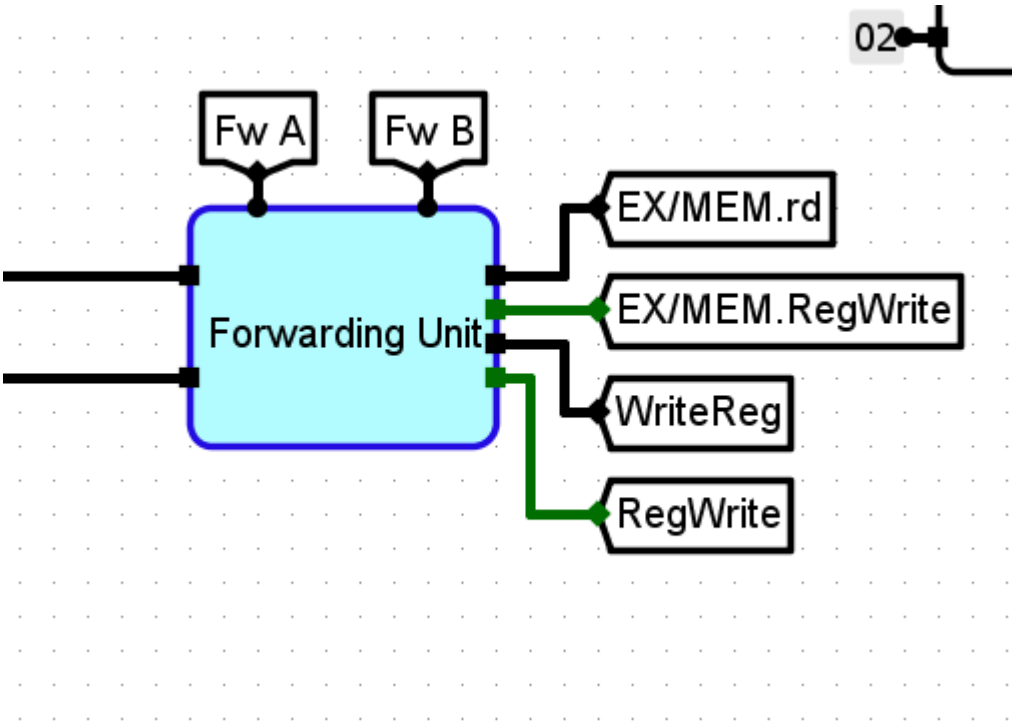


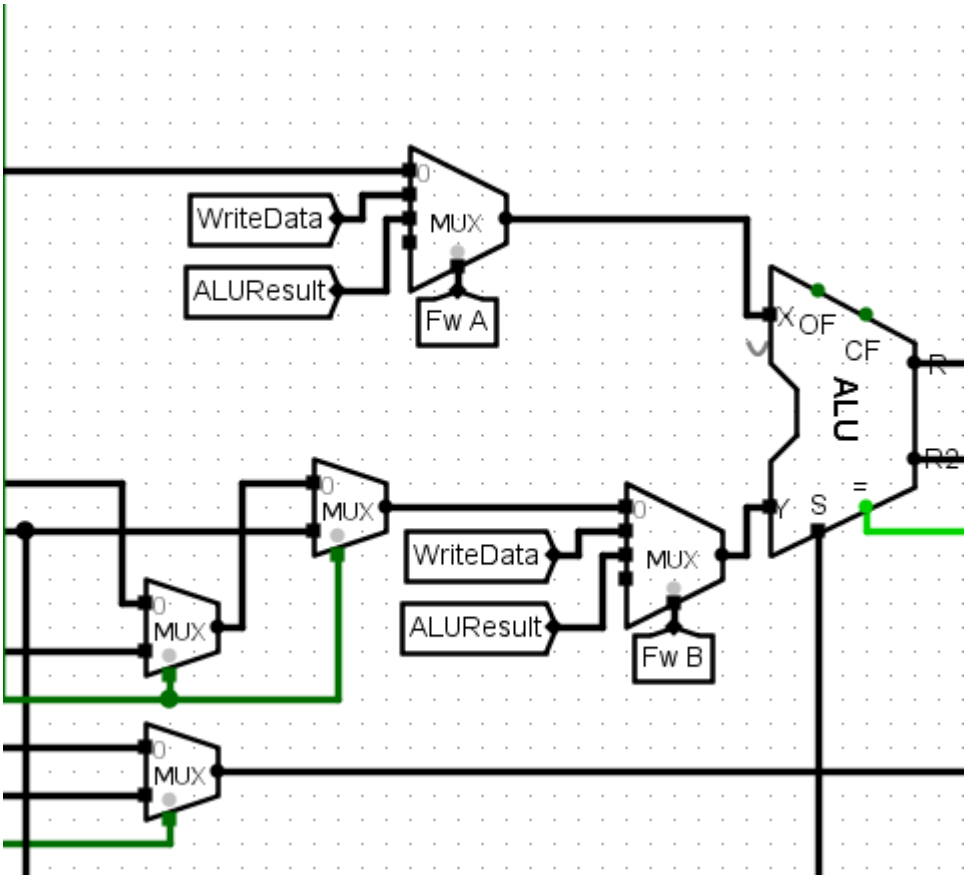
FIGURE 4.57 A close-up of the datapath in Figure 4.54 shows a 2:1 multiplexor, which has been added to select the signed immediate as an ALU input.

Imagen de referencia del Libro

Las salidas del **Forwarding Unit** llegan a modificar al **Src** del ALU. En otras palabras, este módulo incide en que instrucciones o datos ingresan al **ALU**.

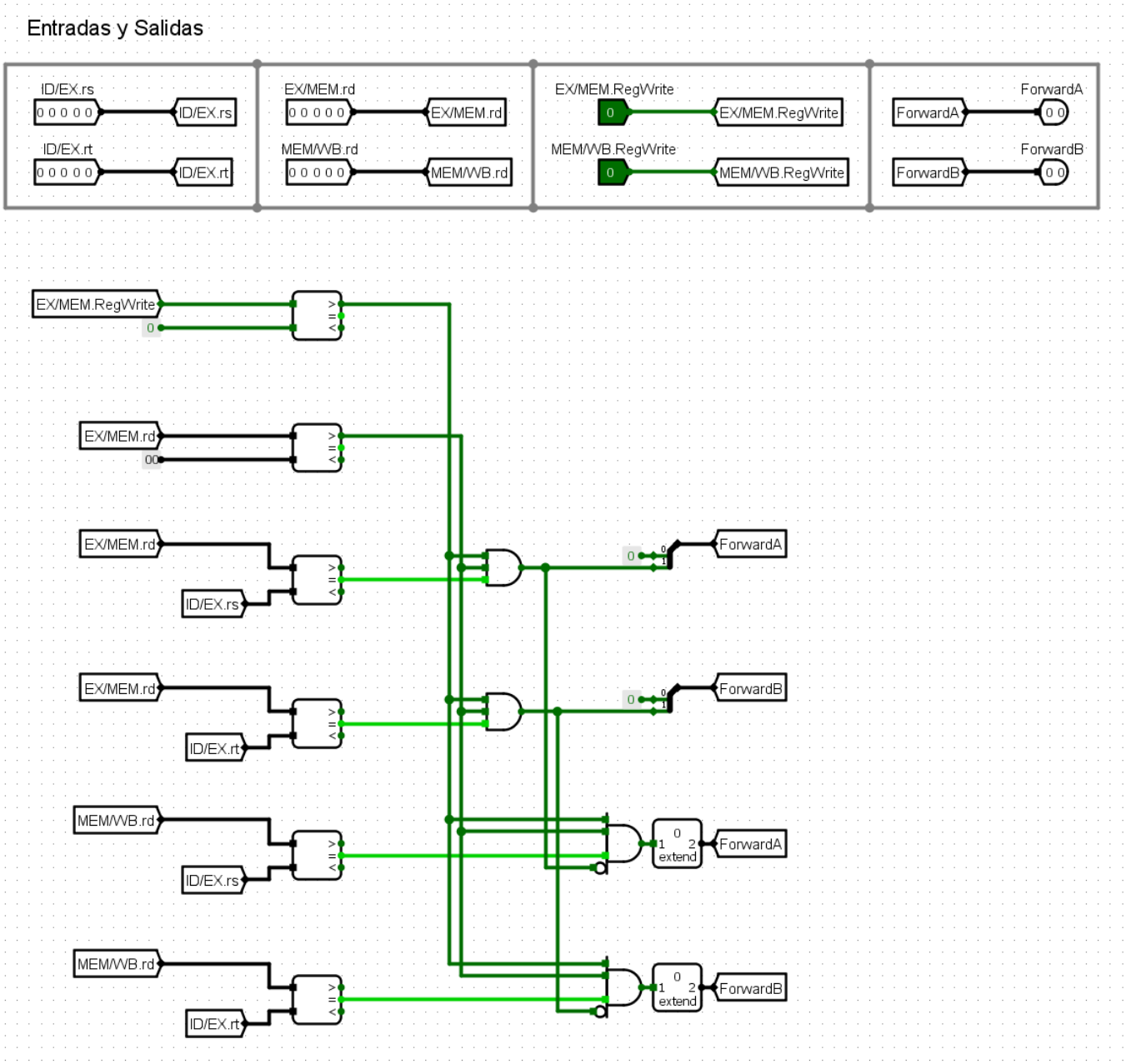


Subcircuito Forwarding Unit



Selectores desde Forwarding Unit

Como se aprecia en la imagen, las salidas **Fw A** y **Fw B** son las que nos permitiran modificar las entradas al **ALU**, funcionando como selectores.



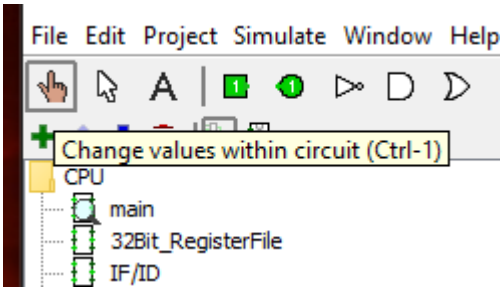
Circuito Forwarding Unit

Las salidas son definidas por condiciones ya establecidas previamente.

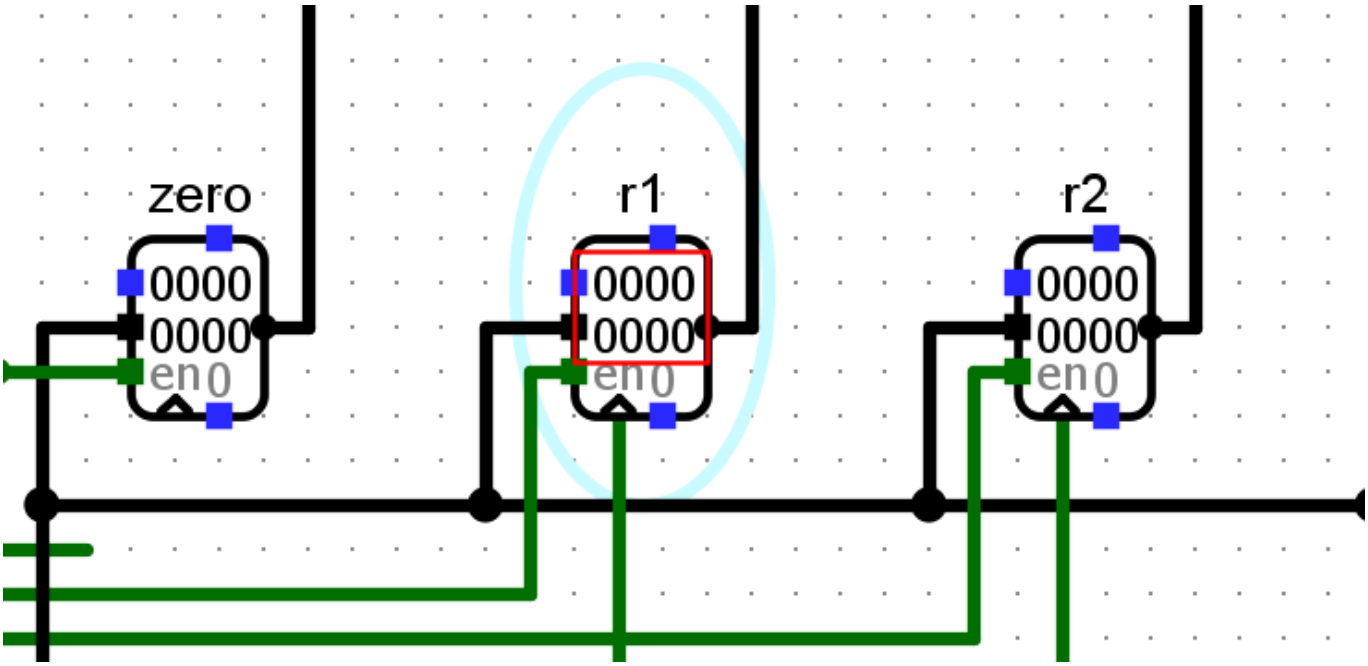
Manual de Usuario

Para correr el ejemplo descrito en la pregunta, se necesita ciertas condiciones previas.

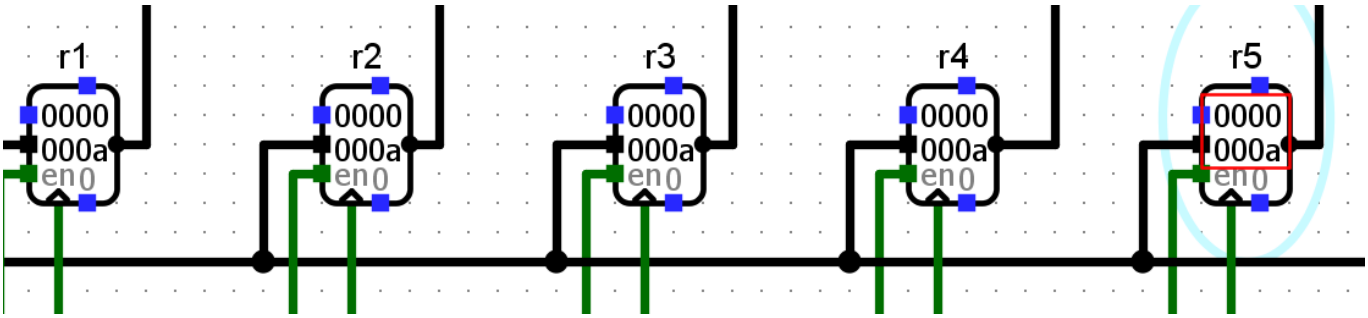
La primera es que debido al diseño, y a las facilidades de logisim, se tiene que ingresar el valor 10 en decimal a los registros del 1 al 10.



1) Escogemos la herramienta de selector

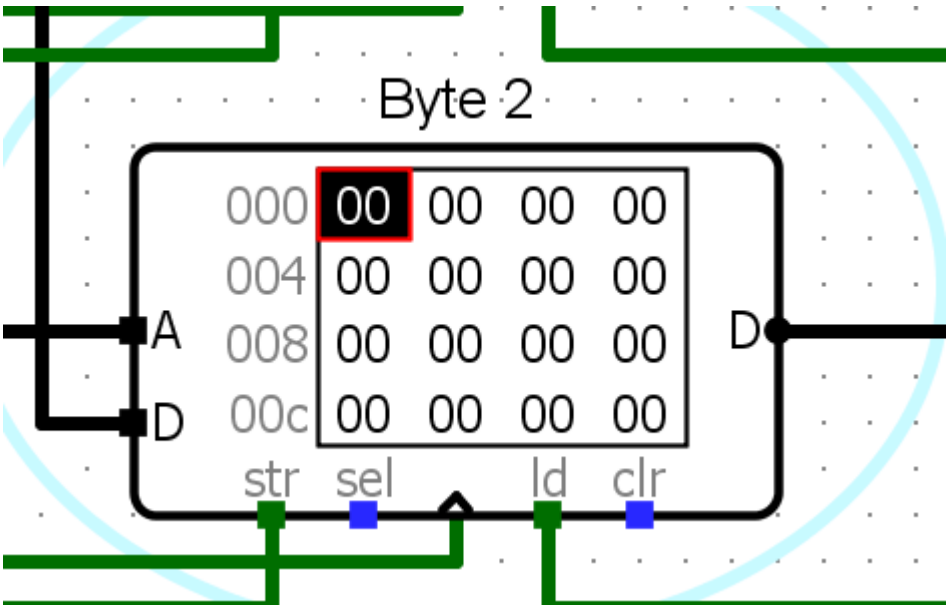


2) Manualmente escogemos los registros del 1 al 6 en el Register File (Ya estan con sus respectivos labels)

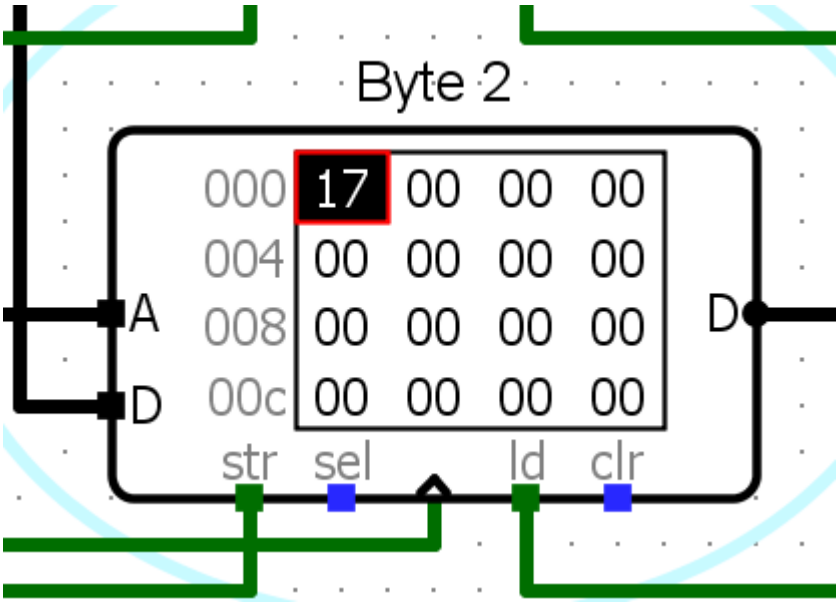


3) Como logisim maneja los registros en Hexadecimal, ingresamos la letra 'a' en los registros del 1 al 6

Adicionalmente, con la misma herramienta tenemos que ingresar el valor 23 en la posición 32 de **Data Memory**



*4) Buscamos y seleccionamos la primera casilla del segundo componente **RAM**. En este caso tiene el label Byte 2*



5) Ingresamos el valor '17' que en hexadecimal es equivalente a 23

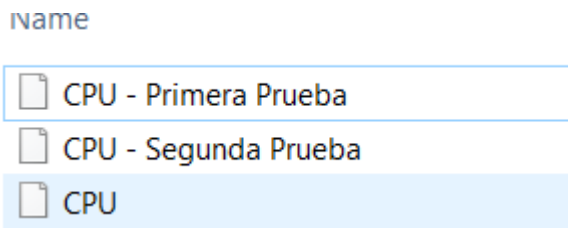
Con estos pasos ya se puede hacer una simulación para el enunciado descrito.

Intrucciones Desensambladas

Adicionalmente, se pide desensamblar estas instrucciones.

```
lw $2, 22($1)     // se carga de la posición de memoria 22 + 10 (tiene un 23)
add $4, $2, $3    // r4 = 23 + 10
add $5, $2, $3    // r5 = 23 + 10
add $6, $4, $2    // r6 = 33 + 23
slt $1, $4, $2    // r4 = 33, r2 = 23, r1 = 0
```

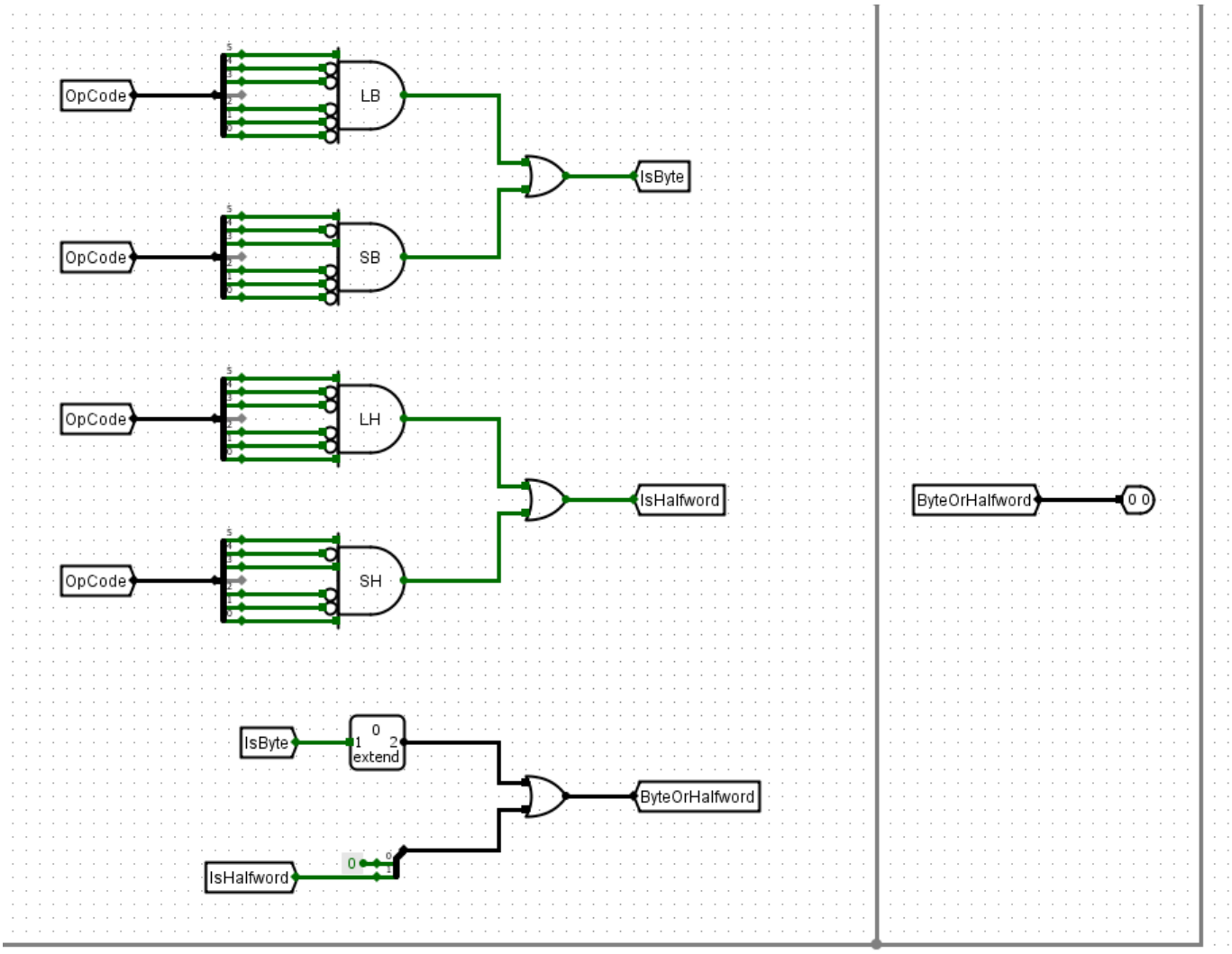
Instruccions a desensamblar



*Los resultados de este cálculo están en el archivo **CPU - Segunda Prueba***

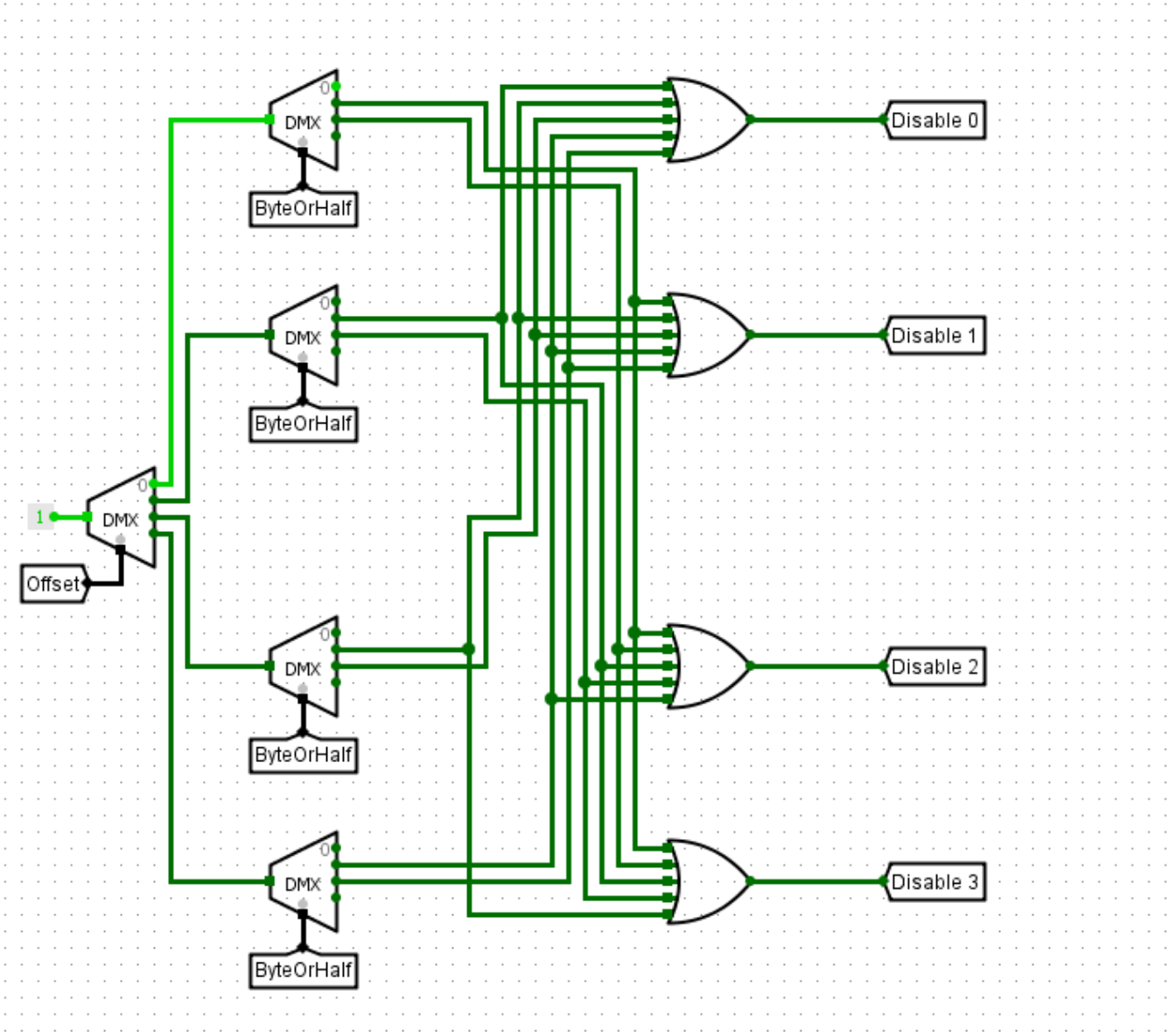
Pregunta B

Para la solución de este problema, se modifico la Unidad de Control para que generara más señales para el Data Memory.



Compuertas lógicas agregadas a la Unidad de Control

Dependiendo del formato de la instrucción, el valor de la salida **ByteOrHalfword** será diferente.

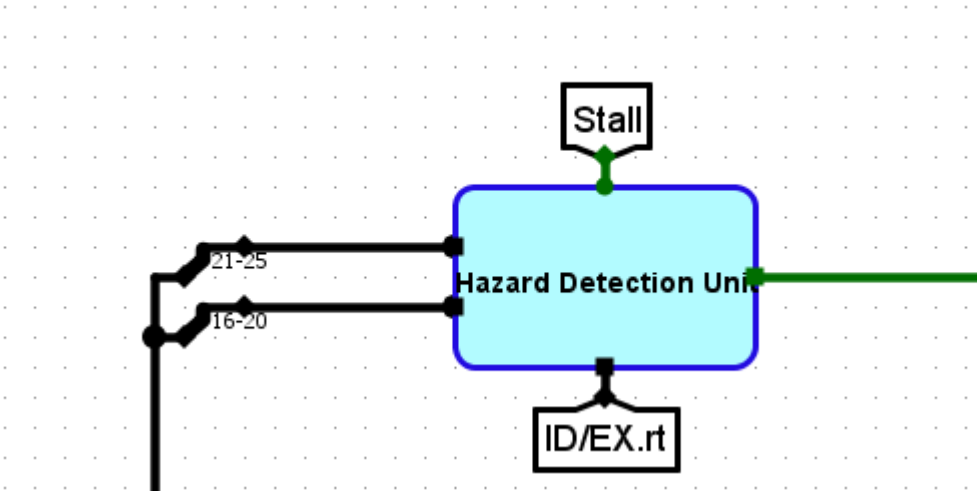


Modificación a DataMemory

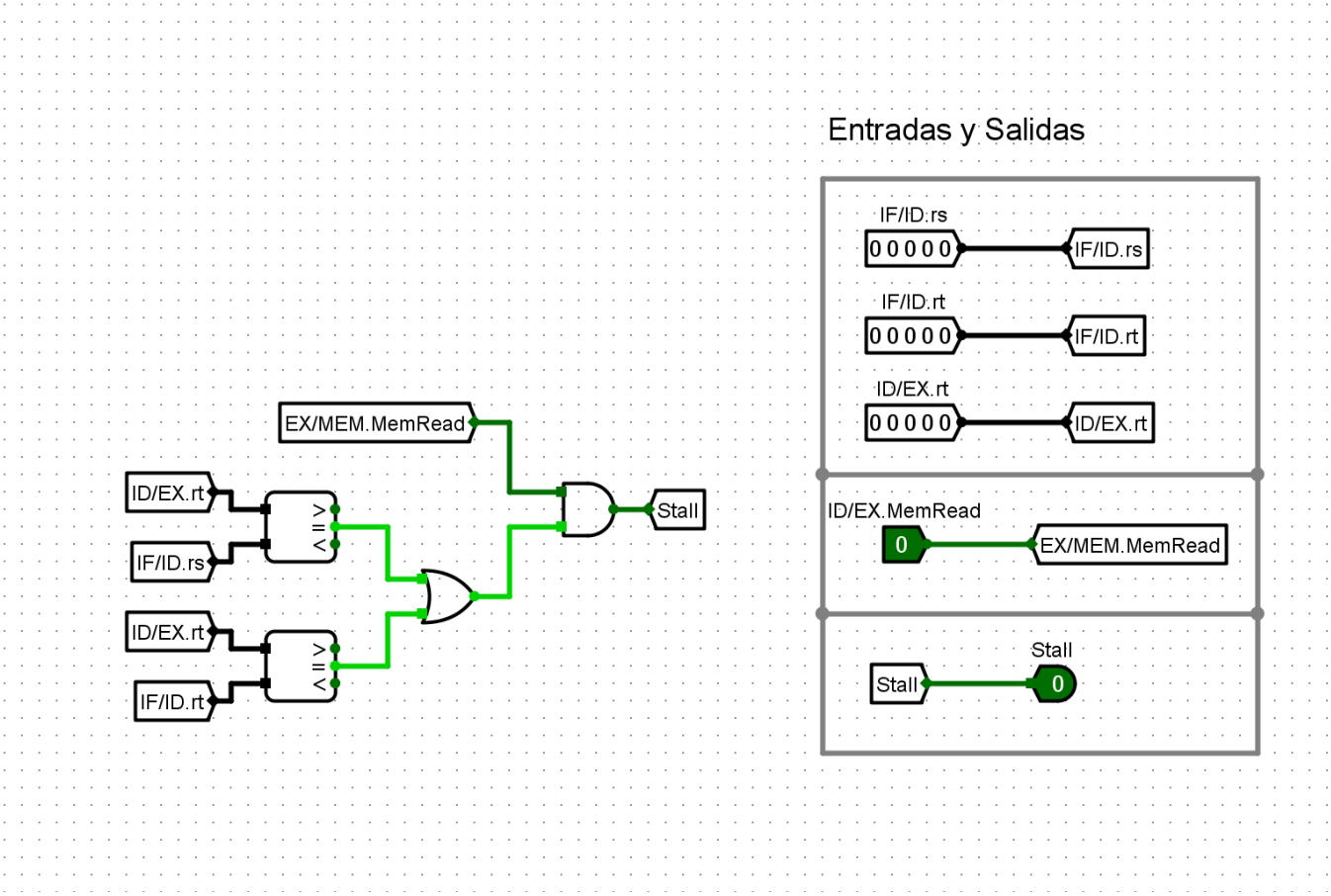
Por medio de este nuevo circuito, agregado a **DataMemory** se desabilitan X cantidad de componentes **RAM**, dependiento de la salida definida en la Unidad de Control. Esto permite interpretar los datos como **Byte, HalfWord o Word**

Pregunta C

Para esta pregunta se escogió implementar una **Hazard Unit**. Sun función primordial es detectar cuando hay conflictos de ejecución entre los stages.



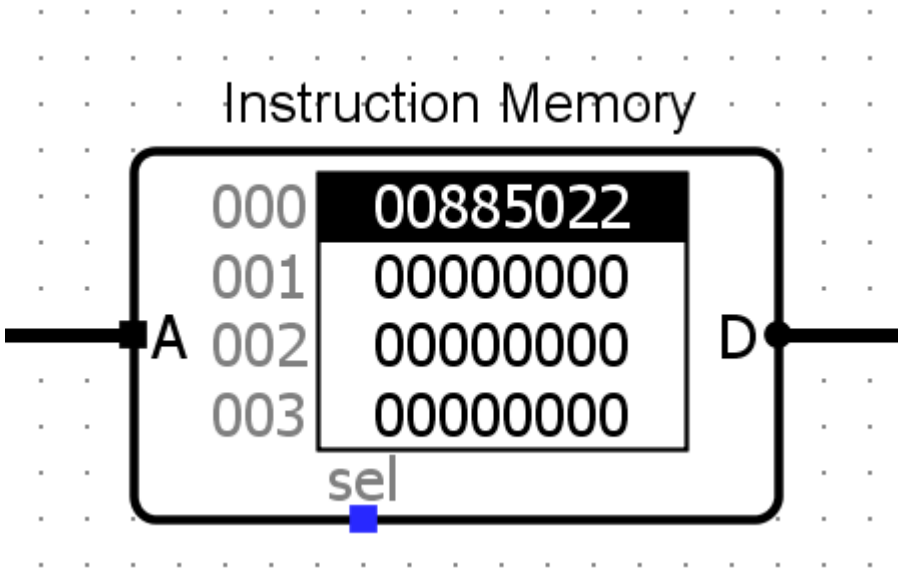
Subcircuito Hazard Unit



Circuito de Hazard Detection Unit

Para este ejemplo, ya la **Instruction Memory** ha sido cargada previamente con un valor de prueba.

Lo que más cabe resaltar de este componente es el flag de **Stall**. Este flag se comunica con los otros stages pertinentes.



Versión de Logisim

Este examen se desarrollo con una versión de **Logisim** que se puede encontrar en este repositorio.