OBJETIVOS

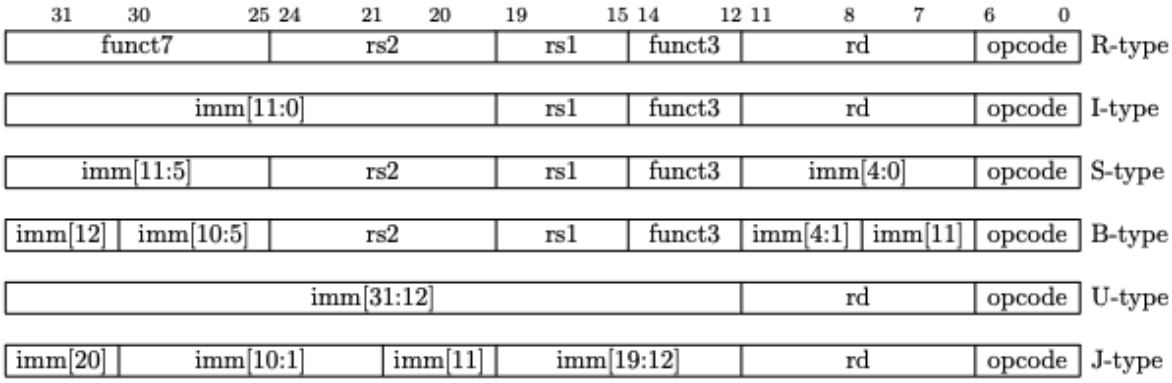
Implementación y diseño de una CPU monociclo en logisim con arquitectura RISC-V la cual debe contener una memoria ROM, una memoria RAM, contador de programa, unidad de control, ALU, y pueda ser capaz de ejecutar correctamente el siguiente set de instrucciones:

Tipo	**Instrucciones**
Aritméticas	ADD, ADDI, SUB
Operaciones lógicas	AND, ANDI, OR, ORI
Bifurcaciones	BEQ, BNE, BLT, BGE
Carga/descarga	LW, SW

Para ello se diseñara una ALU y unidad de control para su utilización en complemento con los componentes de la librería del CS3410.
El siguiente proyecto también se adhiere a las reglas pautadas respecto al diseño simple, la utilización de componentes previstos sin duplicados, y se evitara el uso de demultiplexores.

Previo al diseño y para la correcta implementación de la unidad de control primero se corroboraron las instrucciones salientes de la memoria de programa y con la utilización de splitters se confirmaron todos los diferentes sectores que conforman cada instrucción.

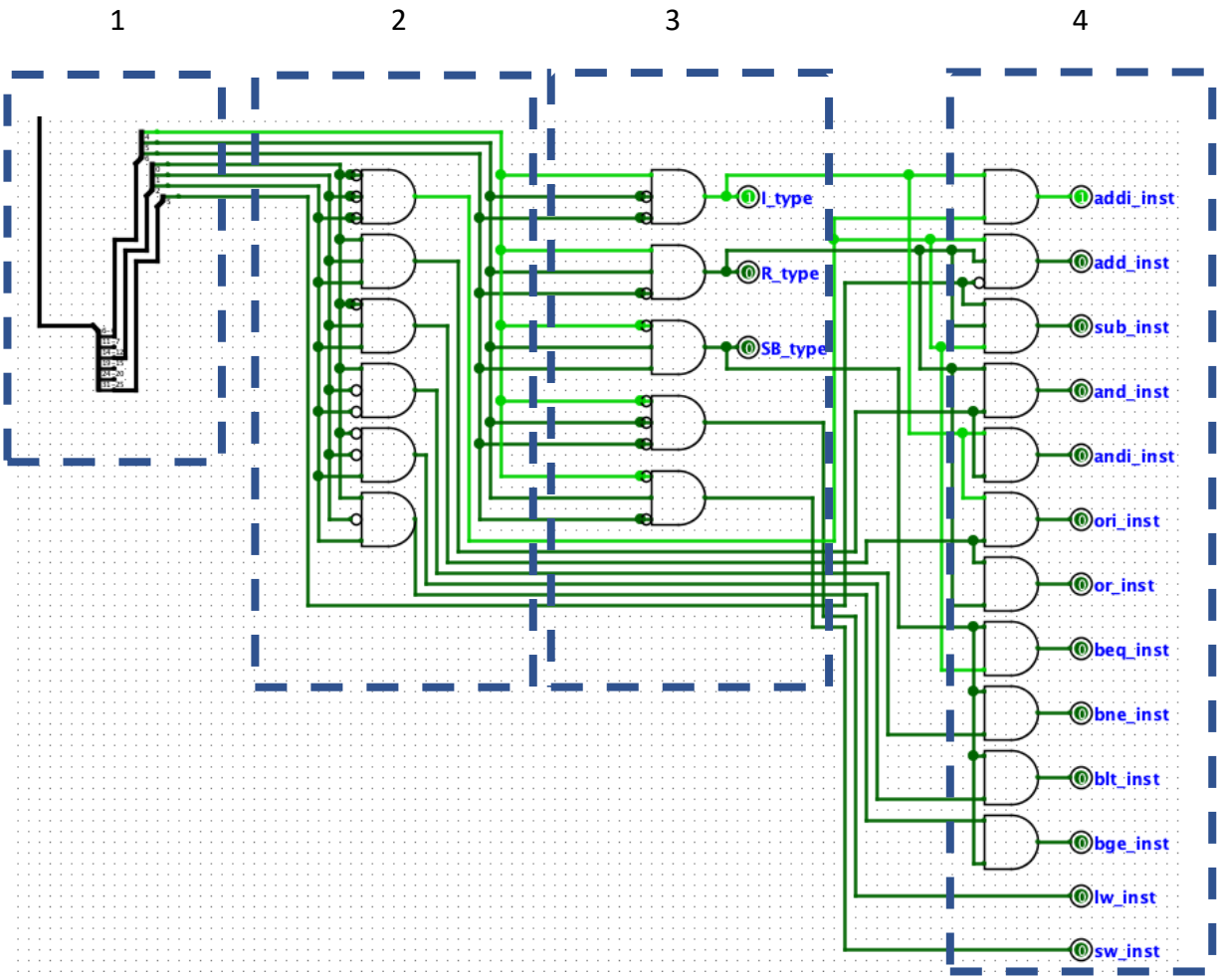
A partir de los datos obtenidos y siguiendo el esquema de instrucciones de RISC-V



Se derivó en la siguiente tabla de verdad para las instrucciones necesarias:

instruction	7bits	5bits	5bits	3bits	5bits	7bits	type
	FUNCT7	RS2	RS1	FUNCT3	RD	OPCODE	
addi	IMMEDIATE		RS1	000	RD	0010011	I-type
andi	IMMEDIATE		RS1	111	RD	0010011	I-type
ori	IMMEDIATE		RS1	110	RD	0010011	I-type
lw	IMMEDIATE		RS1	010	RD	0000011	I-type
add	0000000	RS2	RS1	000	RD	0110011	R-type
sub	0100000	RS2	RS1	000	RD	0110011	R-type
and	0000000	RS2	RS1	111	RD	0110011	R-type
or	0000000	RS2	RS1	110	RD	0110011	R-type
bge	IMMEDIATE	RS2	RS1	101	IMMEDIATE	1100011	B-type
blt	IMMEDIATE	RS2	RS1	100	IMMEDIATE	1100011	B-type
bne	IMMEDIATE	RS2	RS1	001	IMMEDIATE	1100011	B-type
beq	IMMEDIATE	RS2	RS1	000	IMMEDIATE	1100011	B-type
sw	IMMEDIATE	RS2	RS1	010	IMMEDIATE	0100011	S-type

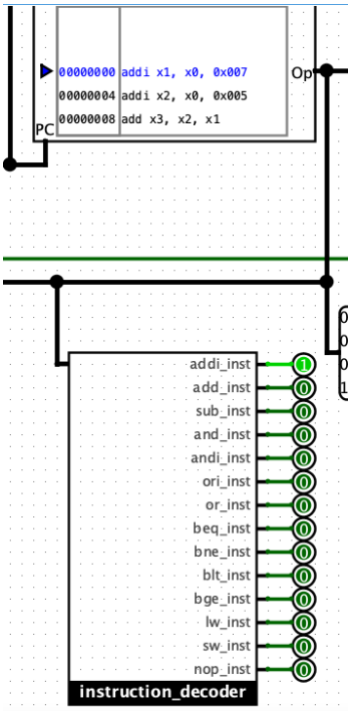
En primera instancia se implemento la decodificación de instrucciones la cual cuenta con 4 etapas y puede determinar la instrucción asi como el tipo:



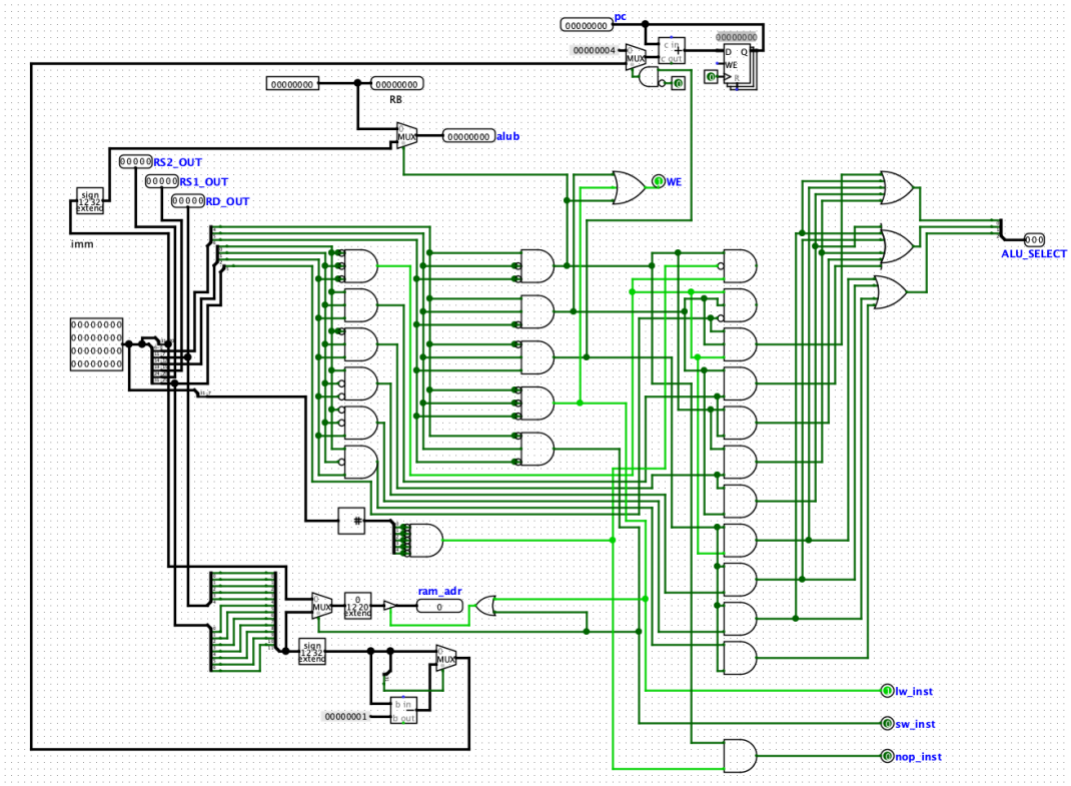
Etapas:

1. Ingresa la instrucción de 32 bits de la cual se extraen el opcode, y func3 o funct7 según necesario.
2. Haciendo análisis combinacional se determinan los bits responsables de definir el tipo de instrucción
3. La etapa 3 se encarga de enviar el tipo de instrucción a la etapa 4
4. Aquí se recibe el tipo de instrucción que en conjunto a los datos generados en la primera etapa pueden definir la instrucción en concreto.

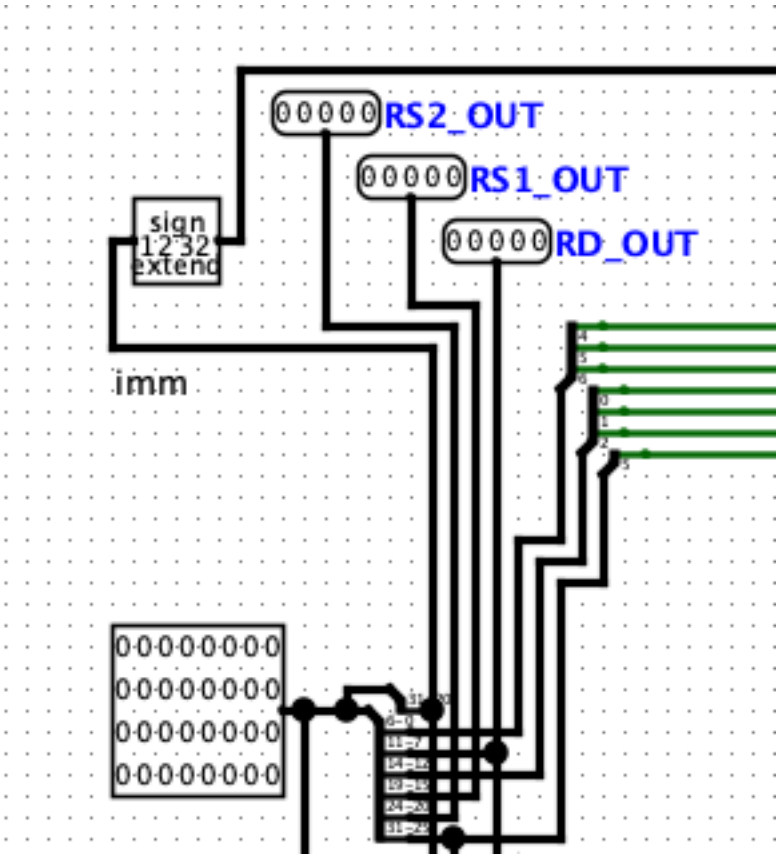
Aquí se puede apreciar el circuito siendo testeado, cabe destacar que no es la forma final de la unidad de control.



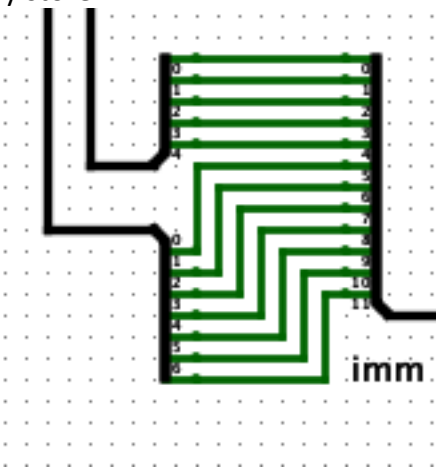
Luego de esto se fueron adicionando los demás circuitos y datapaths necesarios para el manejo del resto de los componentes a la misma. Los cuales se detallan a continuación.



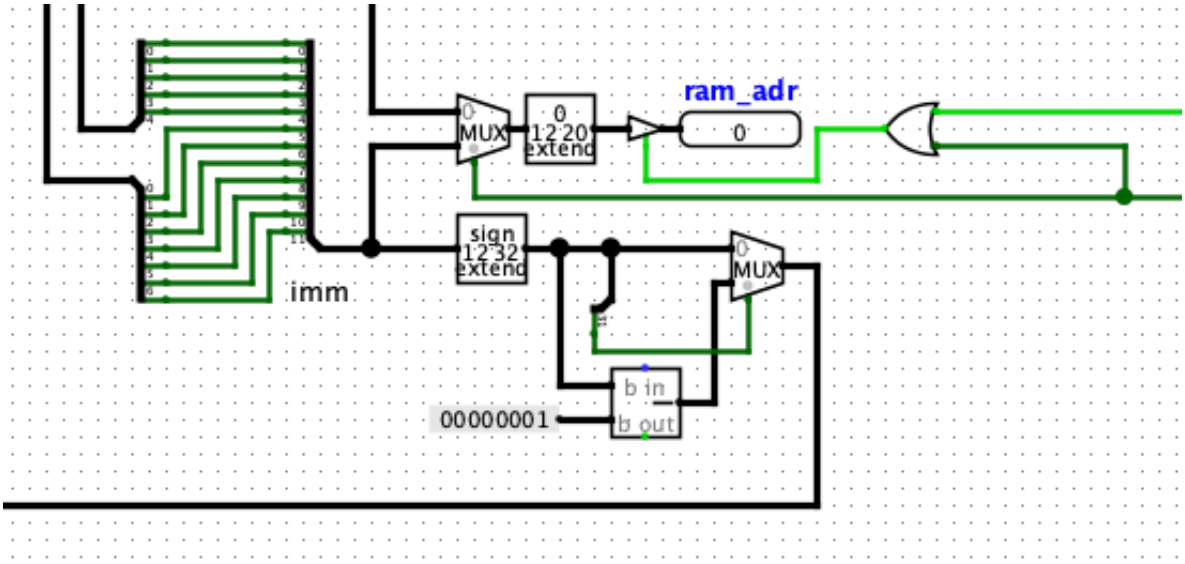
En la esquina superior izquierda podemos observar la generación de las líneas de control para el registro asi como el inmediato para instrucciones tipo I



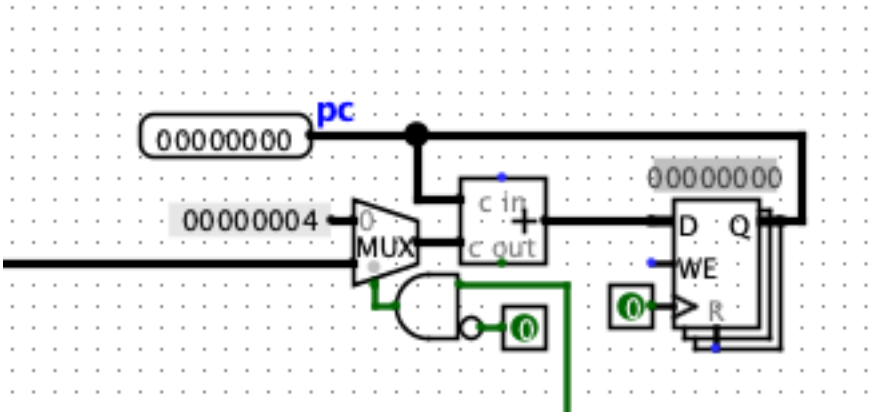
Por debajo de esto se obtiene el otro inmediato de 12bits conformado por los campos RD y funct7 para ser utilizado con instrucciones tipo branch y store



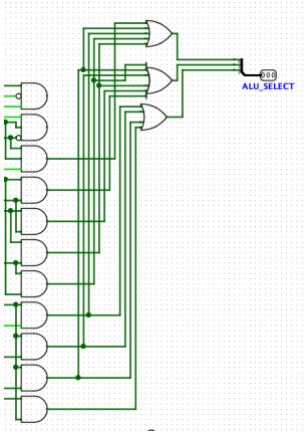
Adyacente al mismo se encuentran ambos circuitos que hacen uso de tal inmediato, en la parte superior se encuentra el multiplexor que decide si para dirección de memoria ram se utiliza el inmediato de las instrucciones tipo I (funct7+rs2) para el load o el inmediato tipo S (funtc7+rd) para el store. La salida de dirección de memoria solo se activa cuando se trate de una instrucción lw o sw.
Por debajo encontramos el circuito que se encarga de calcular el salto de memoria para las instrucciones tipo branch.



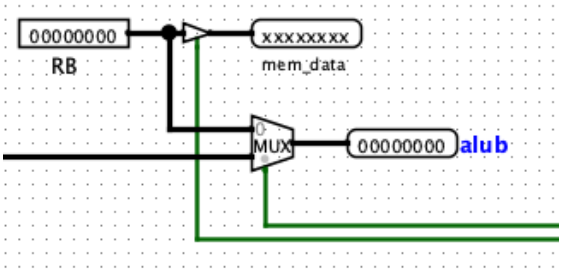
Ya que se hablo del salto de memoria el valor calculado va a nuestro contador de programa ubicado dentro de la unidad de control en la parte superior media, la cual si se trata de una instrucción tipo branch y si la salida de la ALU retorna un 1 cumpliendo asi la condición de salto, modifica la entrada al sumador ingresando el offset de memoria en lugar la constante 4 lo cual ejecuta el salto correspondiente.



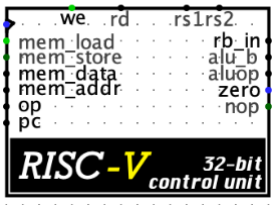
En el sector derecho del circuito de la unidad de control saliendo de la etapa 4 de decodificación de instrucciones utilizamos el mismo método combinacional para generar la señal de control de 3bits para la ALU (se opto por utilizar una señal de 3bits dado que las instrucciones que van a la ALU son exactamente 8, se ampliara este tema en la sección dedicada al diseño de la ALU).



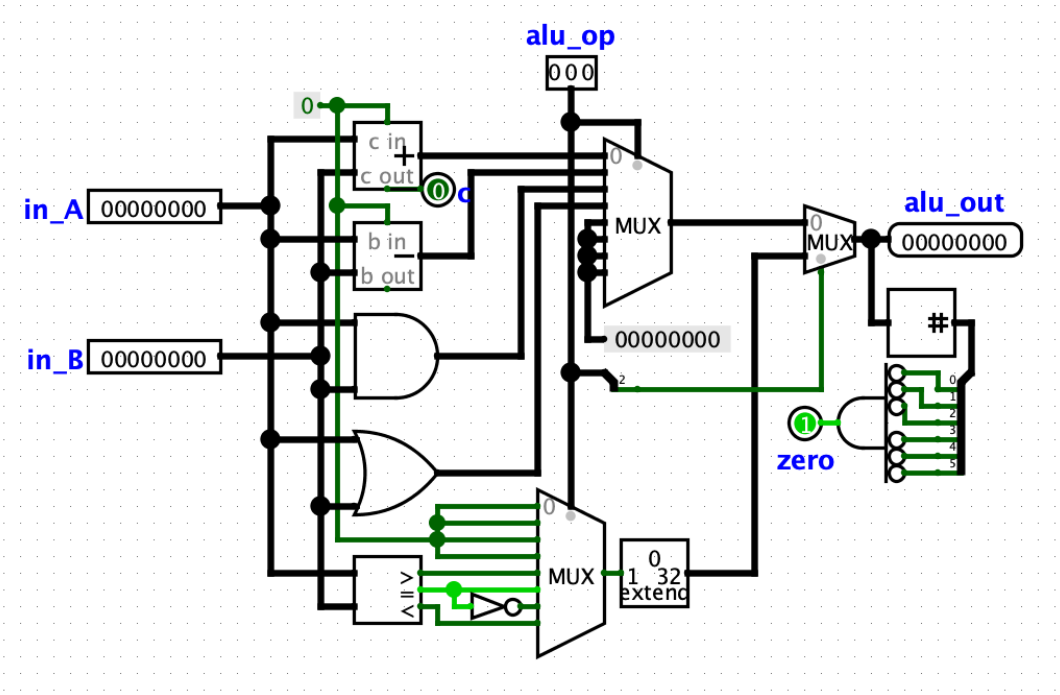
Continuando con el control de la ALU también esta incluido el siguiente circuito que controla cuando enviar como segundo operando de la misma el valor del registro B en caso de tratarse de una instrucción tipo R o pasar el inmediato para las de tipo I. Por sobre la misma hay otro pequeño circuito que envía el contenido de RB como entrada de datos a RAM cuando se ejecuta una instrucción tipo store.



Eso concluye la totalidad de los circuitos en la unidad de control resultando asi el siguiente componente:



Del lado izquierdo se encuentran la entrada de reloj, las líneas de datos y control de RAM como también a la memoria de programa (OP,PC). Del lado derecho se encuentran la salida de control para la ALU, la entrada del registro B y el cero proveniente de la ALU para el control de instrucciones branch. En la parte superior están ubicadas las salidas encargadas de proveer las direcciones para los registros de origen (rs1,rs2), el registro de destino (rd) y el write enable que habilita la escritura del registro cuando se trate de instrucciones que asi lo requieran (tipo I, tipo R).



El circuito de la ALU es relativamente simple, toma dos entradas de 32bits y realiza operaciones de adición, sustracción, AND, OR o de comparación (igual, no igual, mayor o menor). Dado de que se tratan de 8 posibles operaciones se opto por la utilización de 3 bits de control, dos multiplexores con una entrada de control común controlan la salida de la misma (se usaron constantes en las entradas sin usar para no dejarlas flotantes).

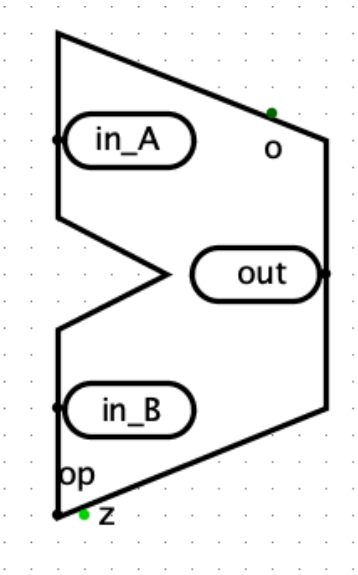
A pesar de que en este especifico proyecto la salida de cero solo se utiliza para controlar la acción de salto de memoria en las instrucciones tipo B, (y bien podría ir a la salida del comparador) se opto por mantener la capacidad de poder determinar el 0 al nivel de salida, por ese motivo la salida del comparador se extiende por 0s a 32bits manteniendo una sola salida de 32 bits controlada por el multiplexor en la salida El mismo cambia de estado con el bit2 de la señal de control.

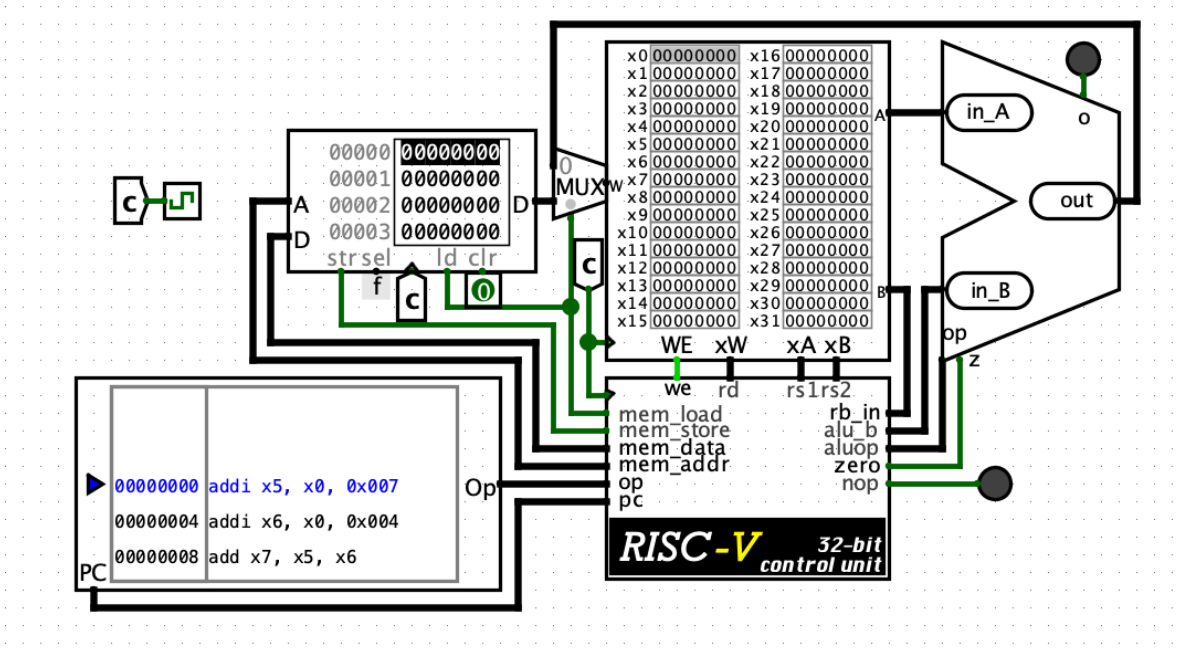
Los opcodes de la ALU se diseñaron de tal manera que se puedan ubicar los componentes con mayor comodidad para mejorar la lectura del circuito.

Esta configuración responde a la siguiente tabla de verdad, con la generación del código de control correspondiente en la unidad de control.

Instrucción	bit2	bit1	bit0
add	0	0	0
sub	0	0	1
and	0	1	0
or	0	1	1
bge	1	0	0
beq	1	0	1
bne	1	1	0
blt	1	1	1

Para una clara identificación dentro del circuito…



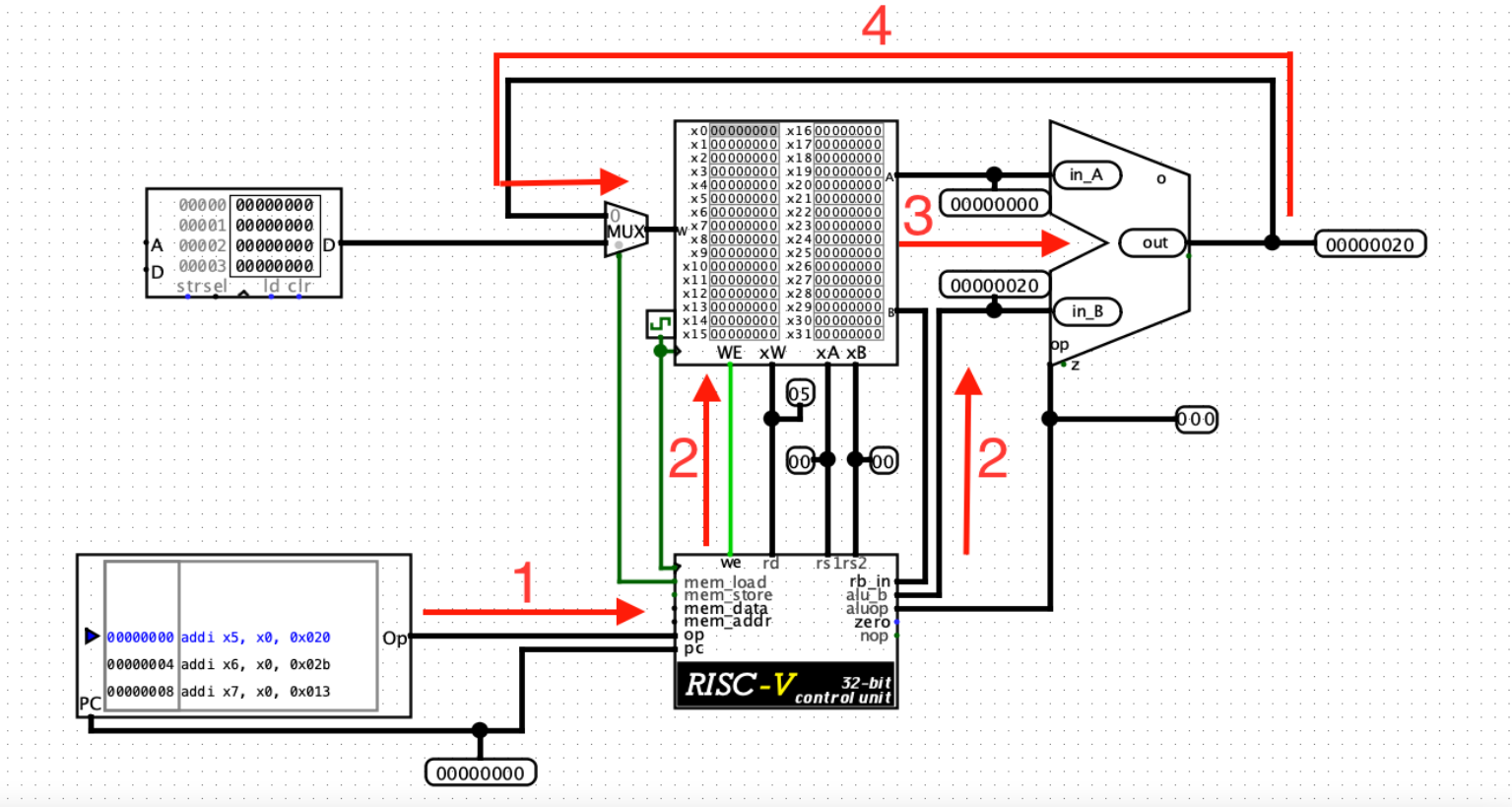


Como se menciono con anterioridad la simpleza y prolijidad para un correcto entendimiento de la misma fueron prioridad en el diseño final logrando asi un circuito pulcro con conexiones claras, sin sacrificar funcionabilidad.

Al tratarse de un set reducido de instrucciones los datapaths son simples, la mayoría de instrucciones salvo por las de load/store hacen uso de la ALU solo teniendo que decidir en estos casos si se trata de una instrucción con un valor inmediato, y un registro o una operación exclusivamente con registros.

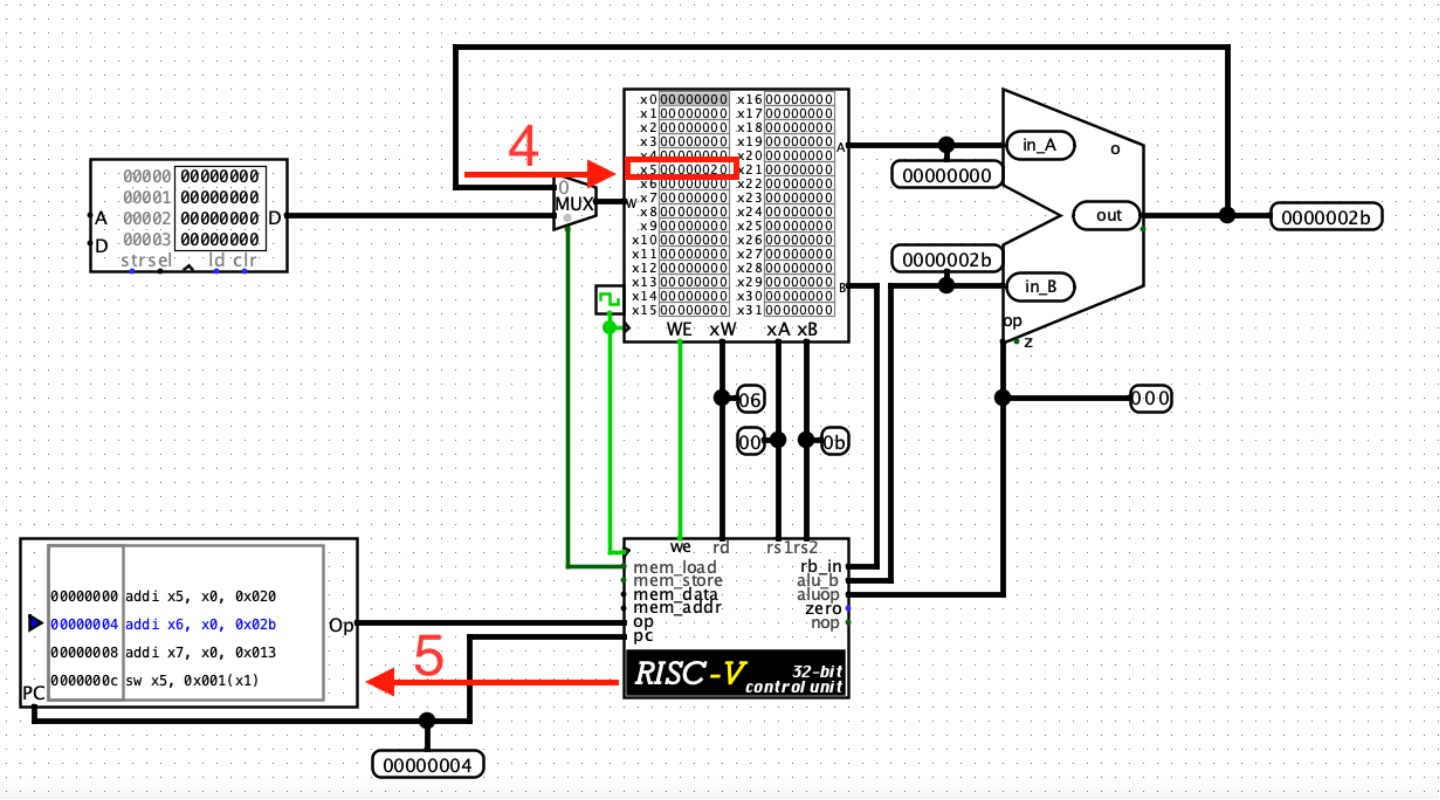
A continuación se ejemplifica el datapath de una instrucción tipo I:

addi x5, x0, 0x020



1. Fetch: la unidad de control recibe una instrucción de 32 bits proveniente de la memoria de programa.
2. Decode: podemos observar que la unidad de control recibe la instrucción, la decodifica e identifica que se trata de una instrucción inmediata de adición con registro de destino 5, y origen 0 con un inmediato 0x020. En consecuencia podemos observar el opcode de ALU 000 correspondiente a la suma, rd mostrando 05 y rs1 00.
3. Execute: La ALU ya tiene en sus entradas el valor del registro x0 junto con el inmediato 0x020 generado por la unidad de control junto con el opcode correspondiente y procede a realizar la suma como podemos observar por el valor en su salida, este valor va al multiplexor de entrada de datos del registro, al no tratarse una instrucción load el valor a escribirse en el registro es el de la salida de la ALU.

4. Writeback: Al pasar al borde ascendente del reloj y tener el write enable activo el registro pasa a almacenar los datos entrantes en W (en este caso nuestro inmediato en la salida de la ALU) en la dirección x5
5. PC+4: Otra cosa que puede observarse es de al no tratarse una instrucción tipo B el contador de programa solo se incrementa en +4 dejando ya preparados los datos y señales de control decodificados de la nueva instrucción en sus salidas.



Finalizando asi un ciclo con una instrucción ejecutada.