OptimusWay 29/04/2022

Miembros:

- Edixon Toro V027364279
- Juan Mora V0
- Gabriel Perez V026587279
- Johan Paredes V027507388

Camino de datos RISC-V

Instruction Fetch/Instruction Decode

Al iniciar la simulación, el programa es cargado a Instruction Memory el cual contiene un array de strings como memoria. Mientras el programa se carga a la memoria, se identifica qué líneas son instrucciones y qué líneas son etiquetas (tags). A partir del primer ciclo de reloj, el Contador de Programa (PC) se inicializa en 0 y empieza a funcionar el procesador, Instrucción memory lee el valor de PC y encuentra la instrucción deseada, como es un dato tipo String, se utilizan métodos para leer, modificar y copiar partes específicas de la instrucción. A pesar de que todas las instrucciones tienen un formato parecido, la mayor dificultad para crear desde cero esta etapa fue implementar las operaciones de Salto, ya que necesitan saber específicamente la dirección a donde saltar. Formato del programa de entrada:

• Si es una instrucción debe estar precedida por un TAB, (‘\t’).

• Si es una etiqueta, no debe estar precedida por ningún carácter y debe terminar con ‘:’ Ejemplo:

```
     add x1, x2, x3
     beq x0, x0, loop
     bne x0, x0, jump
     sub x4, x5, x6
     and x7, x8, x9
     or x0, x0, x0
loop:
     xor x0, x0, x0
     addi x1, x2, 69
     andi x7, x8, 420
     ori x0, x0, 24
     beq x0, x0, exit
     xori x0, x0, 34
     slti x6, x3, 44
     add x1, x2, x3
jump:
     sub x4, x5, x6
     lw x10, 4(x3)
     sw x20, 8(x6)
exit:
```

Esto se encuentra en el .txt

Para Identificar el tipo de Instruccion se creó un diccionario con todas las operaciones

implementadas, llamado **Instructions**

Según el tipo de Instrucción (Operaciones entre registros, Inmediatas, de Carga, de Control, etc)

se llama a una función que leerá los datos de una manera específica para enviar los datos al archivo de

registros y a la unidad de control (TypeR, TypeI, TypeL, TypeB, entre otras más).

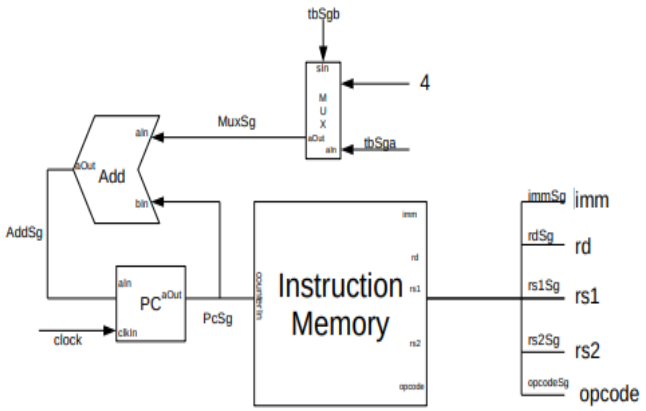
Las señales de salida son:

- **rd:** dirección del registro destino (sc_uint<5>)
- **rs1:** dirección del primer operando (sc_uint<5>)
- **rs2:** dirección del segundo operando (sc_uint<5>)
- **imm:** valor inmediato (sc_int<12>)
- **opcode:** código de operación (sc_uint<5>)

Después de enviar las señales correspondientes, el sumador aumentará el valor de PC en 4 para

leer la siguiente instrucción, o en caso de haber leído una instrucción de salto, el sumador hará una

operación para calcular la dirección de la nueva instrucción que se va a ejecutar



Instruction Memory: Contiene el programa completo a ejecutar, y envía las señales **rd**, **rs1**, **rs2** al archivo de registros; **imm** al sign extend y **opcode** a la unidad de control.

PC (Program Counter): Contiene de la dirección exacta de la instrucción que el procesador va a leer

Add: Calcula la nueva dirección del PC.

MUX: Recibe la señal para seguir el programa secuencialmente (sumar 4 al PC) o hacer un salto.

Figure 1: Diagrama Final

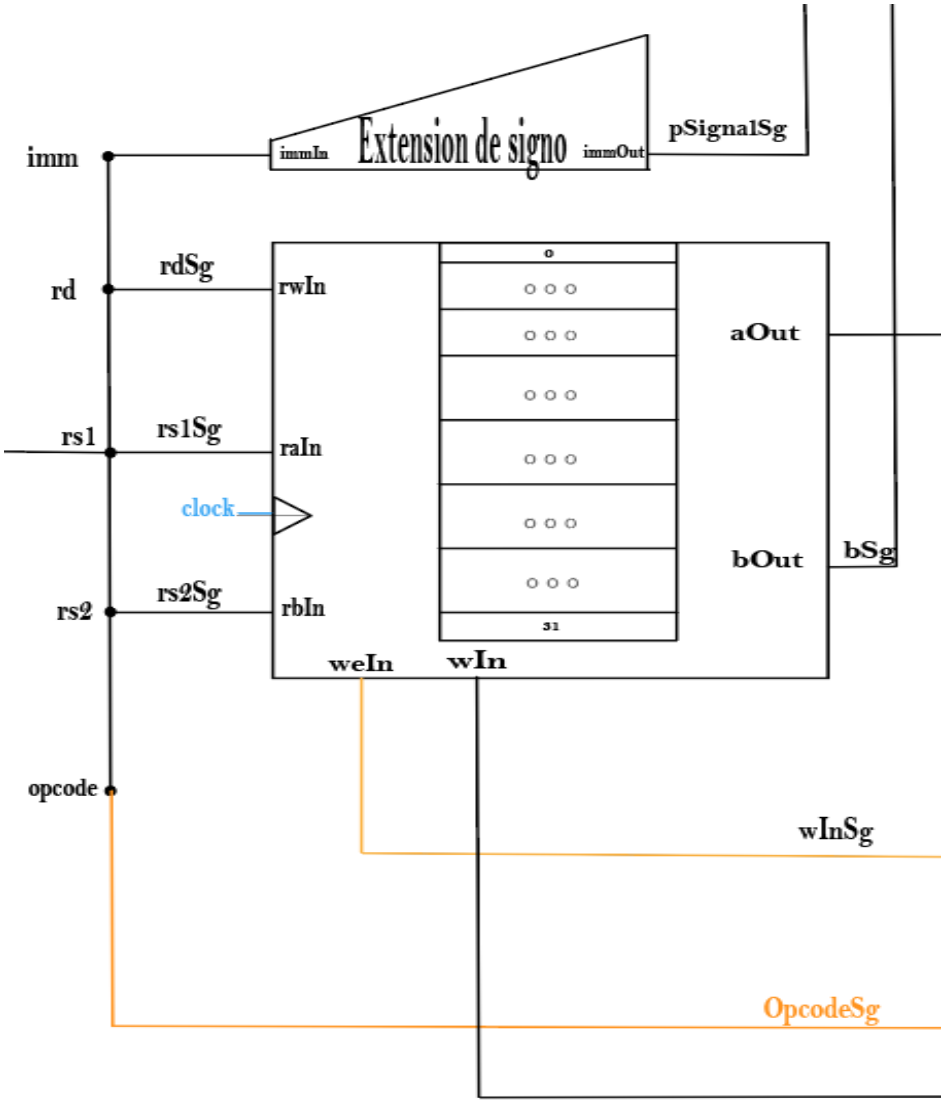
Instruction Decode / Execute

Acá conseguimos una de las secciones que más amplitud tiene respecto a las operaciones a realizar y es el archivo de registro que posee entradas de habilitación de escritura, la entrada de reloj para escribir solo en el flanco de subida, recibe las direcciones desde el Instruction Fetch y le envía los datos según corresponda a la ALU. La composición de este archivo de registro es la siguiente:

- **weIn:** Habilitador de escritura esta señal la recibe de la unidad de control (sc_in<bool>)

- **wIn:** Lo que regresa del Write back (sc_in<sc_uint<32>>)
- **rd:** Direcion de entrada del registro destino (sc_in<sc_uint<5>>)
- **rs1:** Dirección del primer operando (sc_in<sc_uint<5>>)
- **rs2:** Dirección del segundo operando (sc_in<sc_uint<5>>)
- **aOut:** Primer dato salida a la Alu (sc_out<sc_int<32>>)
- **bOut:** segundo dato de salida que se evaluará según corresponda e ira o al multiplexor en EX si se trata de una operación tipo R ira a la ALU, si se trata de una operación tipo I el que pasará a la ALU por este será el valor inmediato(**imm**) o a la MEM si es el caso de un SW (sc_out<sc_int<32>>)
- **clkIn** Entrada del reloj para operaciones respecto a la escritura en el flanco de subida (sc_in<bool>);

- **storage**[31]: Se simula ya datos almacenado en es storage para ver su salida y corregir comportamientos, aca también se guardaran los datos que provengan del WriteBack (sc_int<32>).



Unidad UC

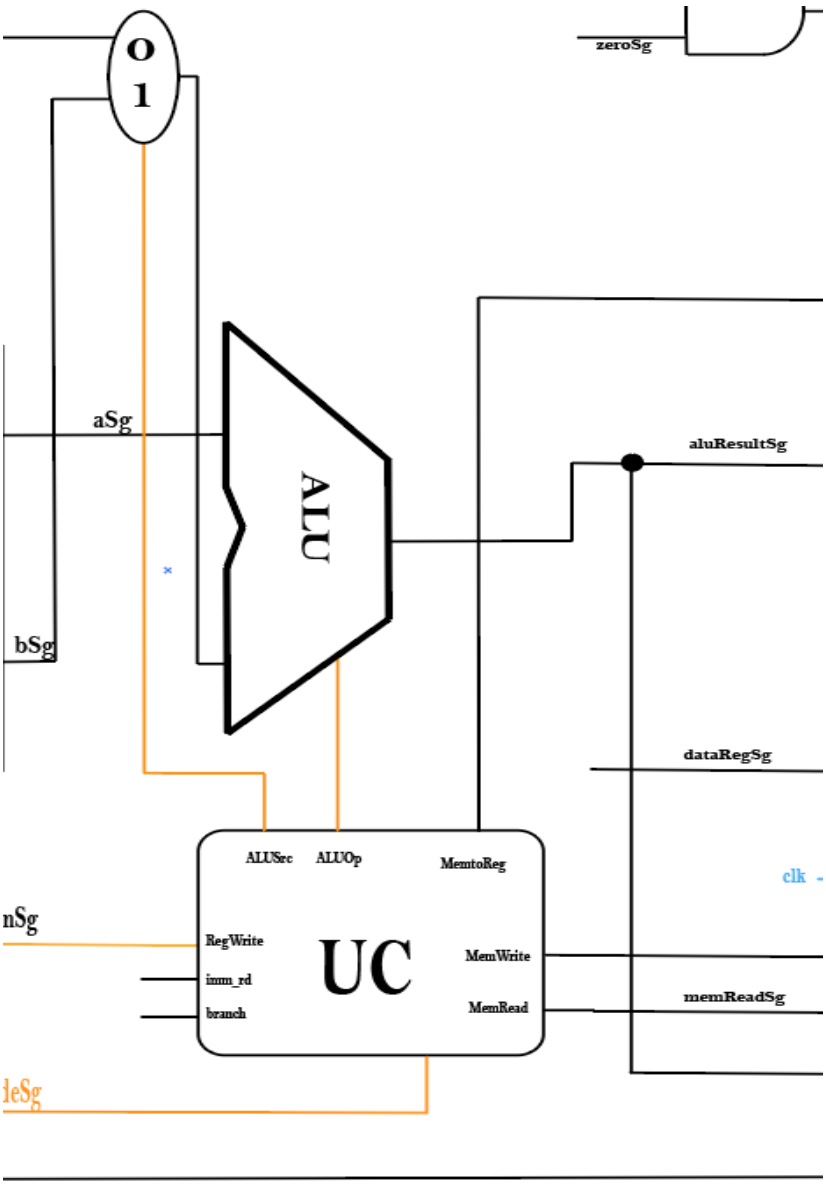
Recibe como entrada el **Opcode** proveniente del archivo de registro el cual indica el tipo e instrucción a realizar para el ciclo de reloj actual. (En el código, **carpeta UC** está los nombres de las señales documentadas). Se usó una constante **enum** llamada instrucción la cual almacena en entero el tipo de instrucción a realizar (desde 0 la cual es INVALID) hasta la instrucción de salto. Si la unidad de control encuentra una instrucción invalida **INVALID** entonces se detendrá la ejecución del programa. El **opcode** se pasa directamente a la ALU por medio de la señal **ALUOpsg.**

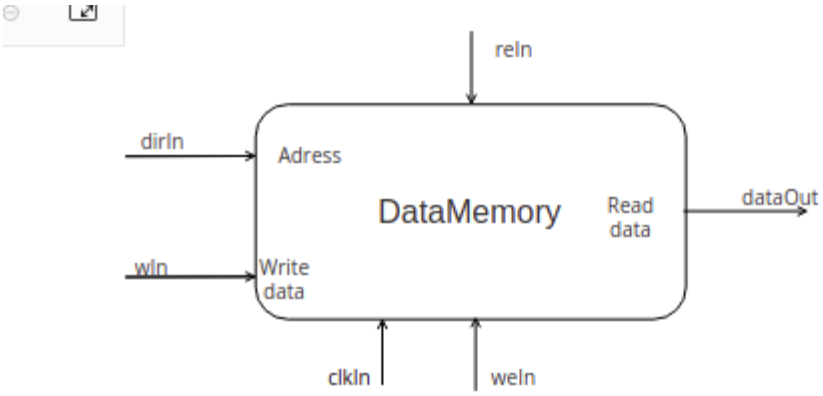
Unidad ALU

Encargada de realizar las operaciones algebraicas y lógicas. Recibe como entrada una señal llamada **ALUOp** de la UC, dos entradas que representa los operandos, uno de ellos se envía de forma directa desde el banco de registro, el otro debe pasar por un multiplexor el cual indicará si debe pasar el valor inmediato (ejemplo para una instrucción de tipo I ADDI) o el segundo registro (una instrucción de tipo R SUB)

Tiene como salida el resultado de la operación que defina el **opcode.** En esta unidad se tuvo el problema (carpeta UC y ALU) que primero pasaba de forma incorrecta el resultado del multiplexor que indica si debe pasar el

valor inmediato o el registro, por tema de circunstancias ajenas no hubo una aplicación para que funcionara para los otros tipos de instrucción.





Leyenda

sc_in<sc_uint<32> > **dirIn** : Direccion en la memoria de datos (Esta direccion es calculada por la ALU)

sc_in<sc_int<32> > **wIn** : Dato a almacenar en la DataMemory

sc_in<sc_int<32> > **dataOut** : Dato Que sera almacenado en algun registro

sc_in<bool > **weIn, reIn** : Señales de la unidad de control si una tiene un valor de 1 la otra debe tener valor de 0 y viceversa

sc_in<bool> **clkIn** : reloj

Memory, se encarga de almacenar datos, está construida como una matriz de n-filas y 8 columnas para simular la forma en que se almacenan los datos que tienen un tamaño de 32 bits, por lo cual cada 4 filas tenemos una palabra. Recibe como entrada una dirección en la DataMemory que es calculada por la ALU normalmente en las instrucciones de store word y load word. Ejemplo si es una instrucción store word. El DataMemory recibe una dirección en la entrada dirIn y también se recibe dato de 32 bits en la entrada Win proveniente de algún registro, luego se ejecuta el proceso de Write que es el que se encarga de escribir los 32 bits en la memoria, básicamente esto se logró recorriendo la matriz desde la posición indicada por la dirección hasta haber recorrido

las 4 filas y en cada posición de la memoria se va almacenando uno de los bits

Ejemplo si es una instrucción load word. Se recibe una dirección en la entrada dirIn, esta dirección proviene de la ALU, no se recibe nada en la entrada wIn porque no es necesaria, a partir de esta dirección se ubica el dato a retornar a alguno de los registros. Esto se implementó de la misma manera en que se explicó anterior mente, solo que en este caso el dato ubicado será enviado a uno de los registros. Cada una de estas instrucciones es controlada por la unidad de control que enviá señales para activar la lectura o escritura en la DataMemory, estas señales son recibidas por las entradas weIn (write enable in) y reIn (read enable in) respectivamente.

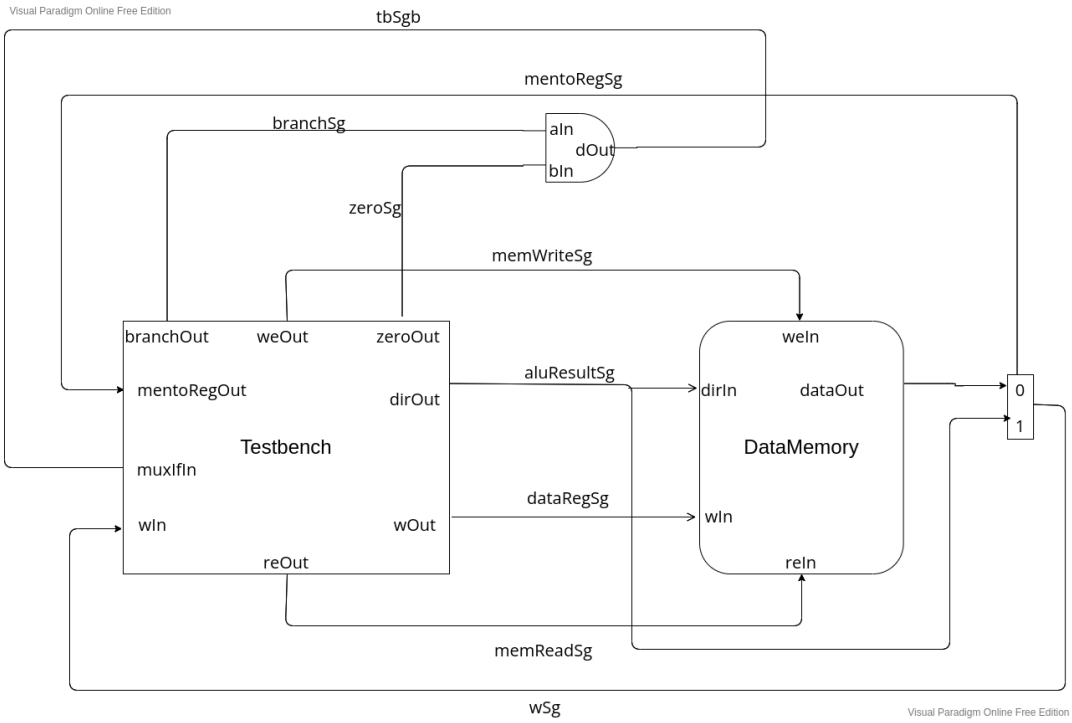
- **MUXWB**

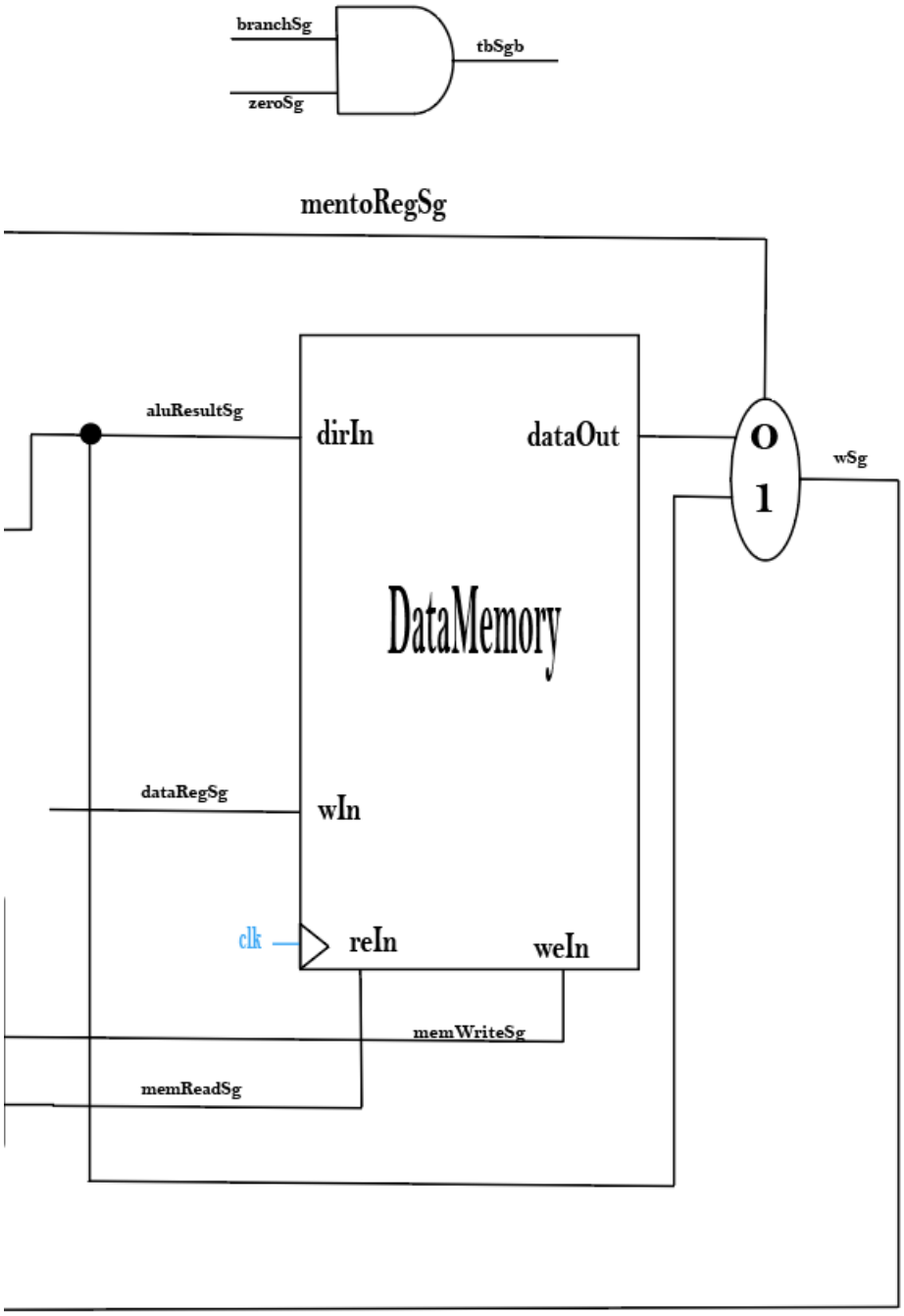
Es simplemente un multiplexor con dos entradas y se encarga de decidir que valor va retornar hacia los registros. Este, contiene dos entras aIn, bIn y una salida cOut, sin mencionar la entrada para la señales de control llamada s0In. La entrada aIn recibe un valor de 32 bits proveniente de la DataMemory, la entrada bIn recibe datos de la ALU, de acuerdo a la señal de control

se decide cual de estos datos recibidos serán retornados a los registros

- **ANDGATE**

Es una compuerta AND, tiene dos entradas aIn y bIn, las entradas provienen de la ALU y de la unidad de control respectivamente, la salida envía el resultado hacia el multiplexor de la etapa IF y es utilizado para determinar si se debe tomar un salto o no. En la siguiente imagen se puede observar un diagrama de la etapa MEM y WB con las conexiones a cada modulo.




branchSg
zeroSg
tbSgb
mentoRegSg
aluResultSg
dirIn
dataOut
O
1
wSg
DataMemory
dataRegSg
wIn
clk
reIn
weIn
memWriteSg
memReadSg

Enlaces

- **https://github.com/optimusway0/RISCV-SIMULATOR/tree/final**
- **https://mega.nz/folder/lJtXGATY#6SpzmRJ9E-SP-HV2HIhq8Q (General todo lo construido, en el comprimido esta lo referente a los test ya probados con ejecución exitosa)**

Para ambos enlaces existe cada sección construida y en el archivo comprimido, será la sección de las etapas IF/ID ID/EX juntas funcionales a partir de un testBench y la etapa MEM/WB funcional con sus respectivas salidas