Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Departamento de computación

Arquitectura de computadoras

CPU: primera entrega

Proyecto

Autor:
André Fabián Castellanos Aldama
Matrícula:
211270011

2022-01-31

Índice

1. Resumen

Se presenta una implementación de una CPU (RV32I) que puede ejecutar las siguientes instrucciones:

1. Tipo-R
2. Load-Store
3. Jump(falta por implementar)

2. Componentes

Siguiendo [1] y las clases de VHDL anteriores. Producimos los siguientes componentes

2.1. ALU (unidad de lógica aritmética)

La ALU se encarga de hacer las operaciones aritméticas del procesador como adición, multiplicación, operaciones lógicas a nivel de bits y en nuestro caso tiene shifts (aunque algunas otras implementaciones dejan los shifts para otro componente que se conoce como shifter).

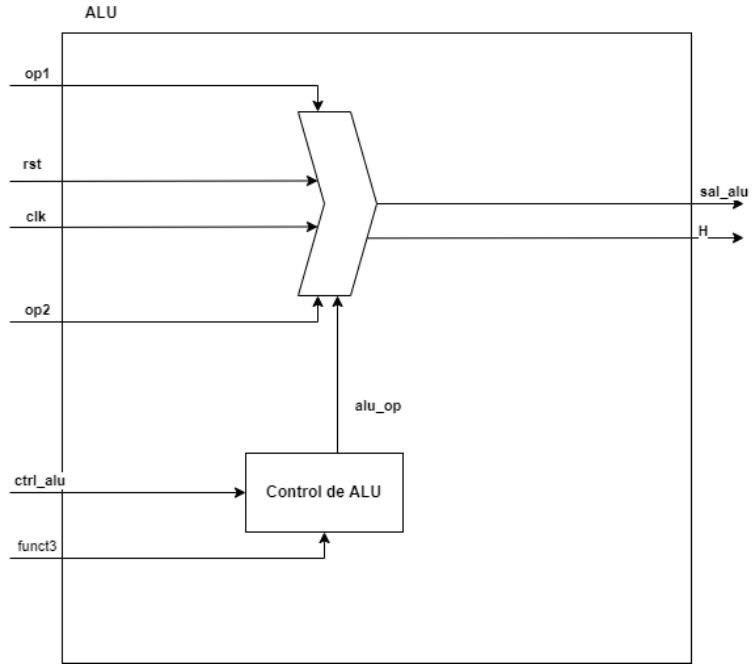

Figura 1: Diagrama de la ALU.

La entrada *ctrl_alu* controla las operaciones que han de hacerse sobre op1 y op2, aunque en mi caso puse la funct 3, en realidad no lo uso.

2.2. PC (contador de programa en inglés)

El PC se encarga de dar como salida la siguiente dirección de la instrucción a ejecutar, tiene un registro interno donde guarda la dirección. Normalmente salta de palabra en palabra, pero también se puede comandar un salto a otras direcciones de ± 1 Byte (según [2]).

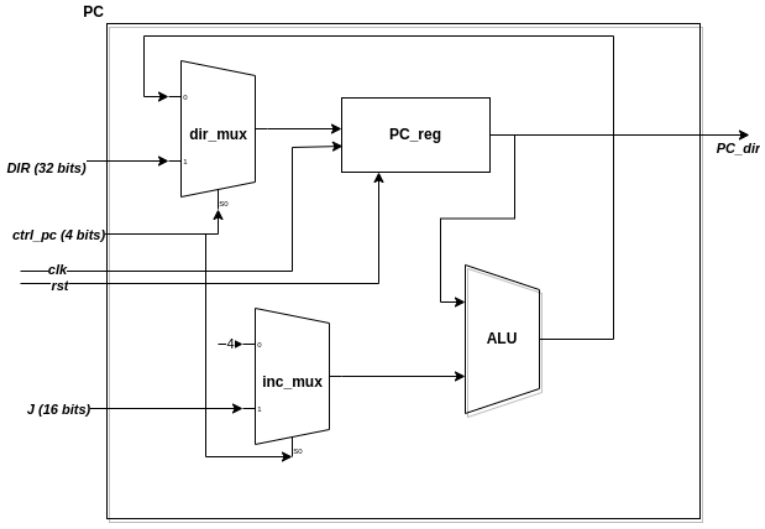

Figura 2: Diagrama del PC.

En nuestro caso el PC salta hacia adelante para esta entrega aunque puede implementarse haciendo que tome en cuenta la entrada *J* como una señal con signo.

2.3. Banco de registros

El banco de registros para el caso de RV32I tiene 32 registros de 32 bits cada uno. Este componente puede leer 2 registros y escribir solo a 1 registro.
Las entradas *reg_write* y *we* nos permiten controlar la escritura y lectura. Para RV32I, el registro ***x0*** es siempre 0 [2].

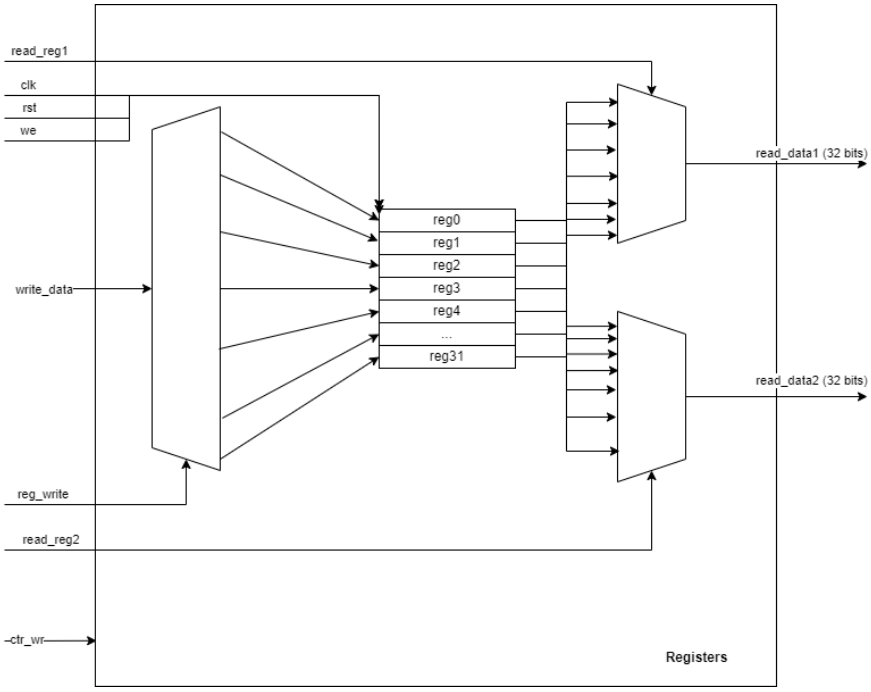

Figura 3: Diagrama del banco de registros.

2.4. RAM

La memoria RAM es una memoria más grande que el banco de registros, aunque solo se puede escribir y leer una dirección.

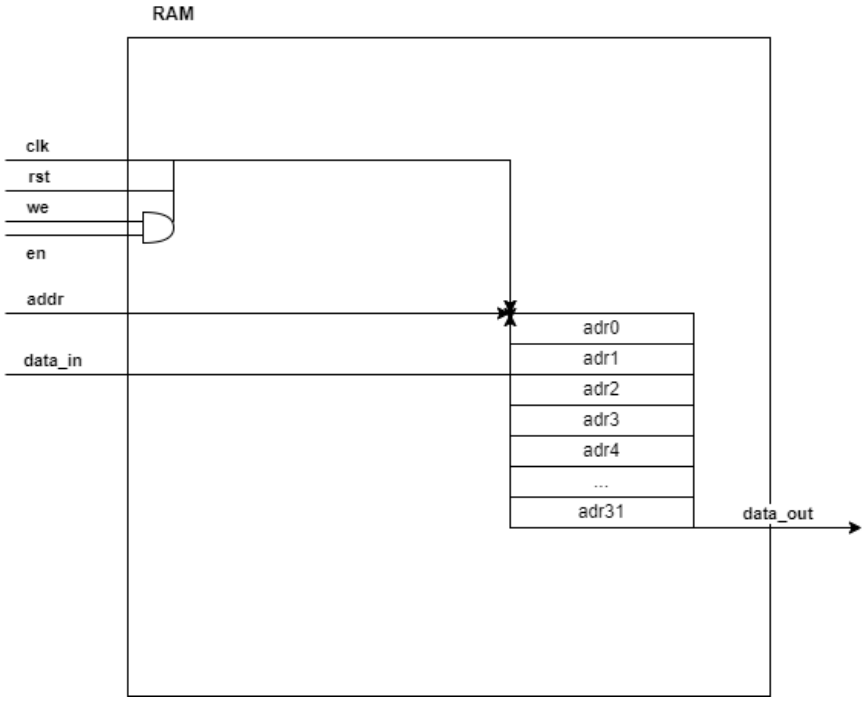

Figura 4: Diagrama de la RAM.

2.5. Immediate generator

El immediate generator lee la instrucción completa para pasar directamente a la ALU operaciones a hacerse para así solo usar los datos de un solo registro o ninguno (para el caso de load).

2.6. Instruction memory (Memoria de instrucciones)

Esta memoria es solo de lectura (ROM) y es el que el contador va ir direccionando para manejar el flujo de ejecución.

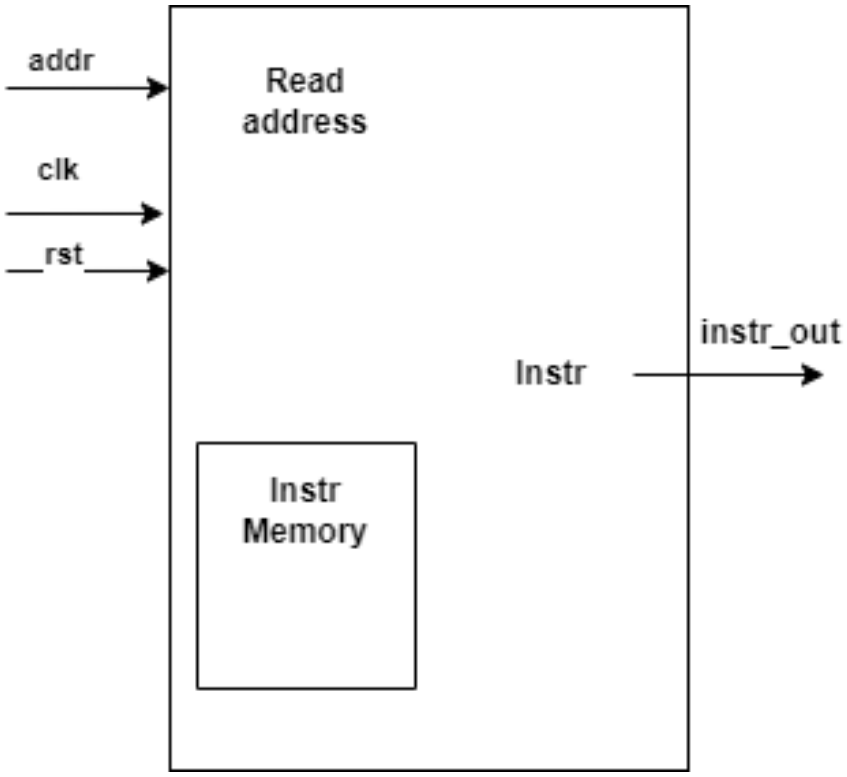

Figura 5: Diagrama de la memoria de instrucciones.

3. CPU

Para implementar el CPU usaremos el diagrama que se provee en [1].

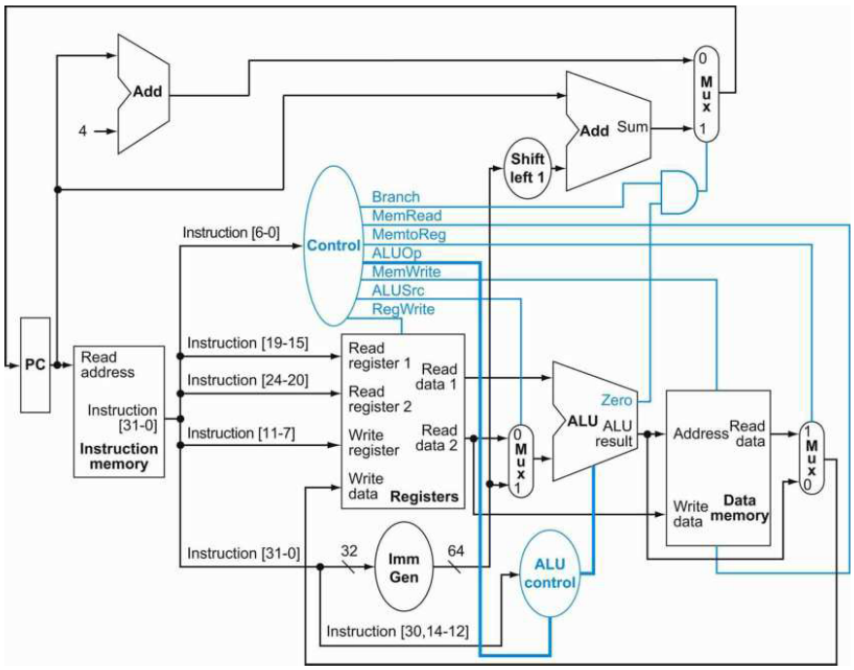

Figura 6: Diagrama de implementación.

4. Pruebas de ejecución de CPU

En este caso cargamos la memoria de instrucciones, que para esta entrega son las siguientes:

```
type mem_type is array(18 downto 0) of std_logic_vector(n-1 downto 0);
    constant ROM: mem_type:= (
        "00000000001000001000001000000001", -- sub x4 x1 x2
        "00000000011100000100011000001100", -- store 7 x4 0
        "00000000001000001000001000000001", -- sub x4 x1 x2
        "00000000001100001000001000000010", -- mult x4 x1 x3
        "00000000001100010000000010000000", -- add  x1 x2 x3
        "00000000000000001100100010001011", -- load x31 3 0
        "00000000000000001001000010001011", -- load x30 1 0
        "00000000000000010100100011001011", -- load x29 5 0
        "00000000000000010100100011001011", -- load x28 5 0
        "00000000000000001100100010001011", -- load x27 3 0
        "00000000000000001001000010001011", -- load x26 1 0
        "00000000000000010100100011001011", -- load x25 5 0
        "00000000000000010100100011001011", -- load x24 5 0
        "00000000000000001100100010001011", -- load x23 3 0
        "00000000000000001001000010001011", -- load x22 1 0
        "00000000000000010100100011001011", -- load x21 5 0
        "00000000000000010100100011001011", -- load x20 5 0
        "00000000000000001100100010001011", -- load x19 3 0
        "00000000000000001001000010001011", -- load x18 1 0
        "00000000000000010100100011001011", -- load x17 5 0
        "00000000000000010100100011001011", -- load x16 5 0
        "00000000000000001100100010001011", -- load x15 3 0
        "00000000000000001001000010001011", -- load x14 1 0
        "00000000000000010100100011001011", -- load x13 5 0
        "00000000000000010100100011001011", -- load x12 5 0
        "00000000000000001100100010001011", -- load x11 3 0
        "00000000000000001001000010001011", -- load x10 1 0
        "00000000000000010100101001001011", -- load x9 5 0
        "00000000000000001100101000001011", -- load x8 3 0
        "00000000000000001001001110001011", -- load x7 1 0
```

```
        "00000000000000101001001100001011", -- load x6 5 0
        "00000000000000011001001010001011", -- load x5 3 0
        "00000000000000001001001000001011", -- load x4 1 0
        "00000000000000101001000110001011", -- load x3 5 0
        "00000000000000011001000100001011", -- load x2 3 0
        "00000000000000001001000010001011" -- load x1 1 0
    );
```

Listing 1: Memoria de instrucciones

Las instrucciones en memoria generan satisfactoriamente muchas de las instrucciones. Hacen falta mas pruebas, pero en general el testbench. Los testbench tanto del CPU como de cada componente se encuentra en el siguiente link.