Práctica 1

Estructura de Computadores

Miguel Lopes
Oriol Poblet
Curs 2022/2023

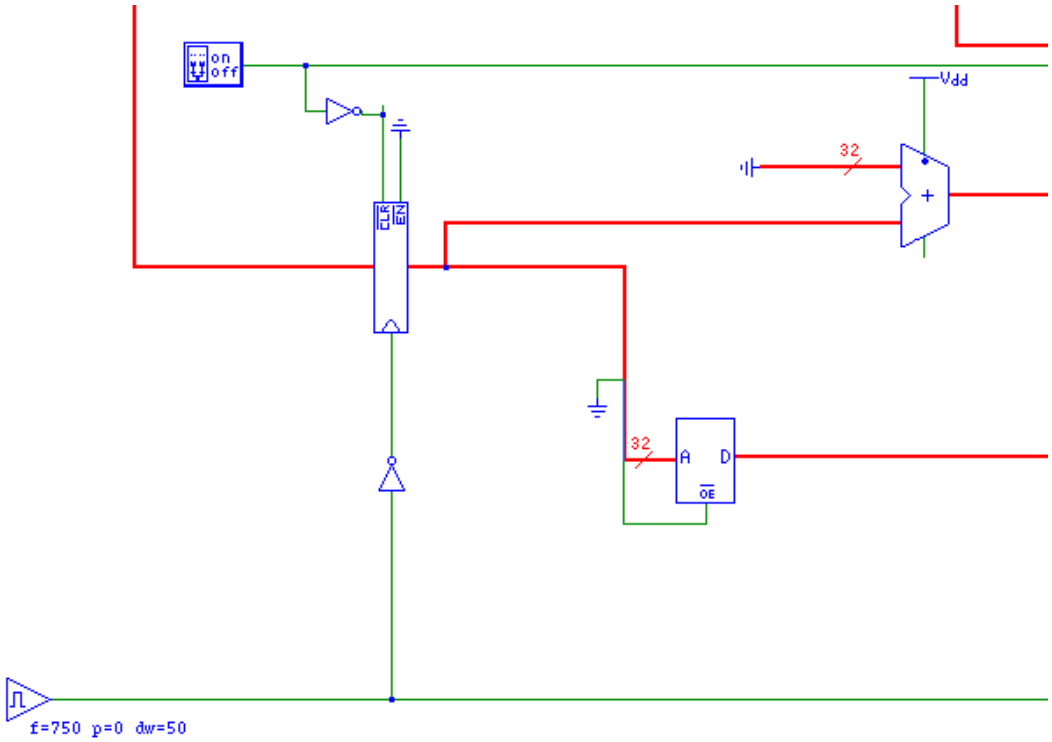
Índex

Especificaciones

TAREA 1. Realizad la parte del circuito del procesador que se encarga del fetch de las instrucciones y que se muestra en la siguiente figura.

En un procesador monociclo, el proceso de fetch de instrucciones implica obtener una instrucción desde la memoria principal y almacenarla en el registro de instrucciones del procesador. El proceso comienza cargando la dirección de la próxima instrucción en el registro de contador de programa (PC). Posteriormente, esta dirección se utiliza para acceder a la memoria principal y recuperar la instrucción.

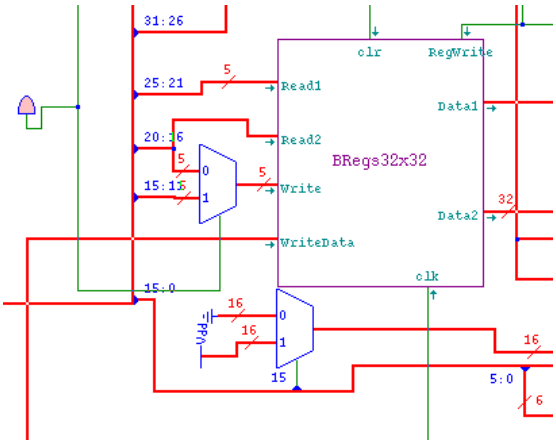
En el caso de un procesador monociclo, la instrucción se obtiene y almacena en el registro de instrucciones en un solo ciclo de reloj.



TAREA 2. Utilizando el banco de registros que se proporciona a través del Campus Virtual, probad el circuito que implementa la etapa de lectura de los registros y que se muestra en la siguiente figura. Realizad varios ciclos de escritura y lectura para comprobar el correcto funcionamiento del mismo.

El Read se divide en dos componentes: el Banco de Registros y la Extensión de Signo. El Banco de Registros utiliza dos direcciones de 5 bits para elegir los registros fuente y dos salidas para obtener los valores de esos registros. También tiene una entrada para seleccionar el registro de destino para almacenar los datos. La Extensión de Signo se utiliza solo para instrucciones de formato I, como "lw" o "sw", para extender los 16 bits de la dirección de memoria a 32 bits con la replicación del bit de signo.

Debemos realizar varios ciclos de escritura para comprobar el funcionamiento del banco de registros.

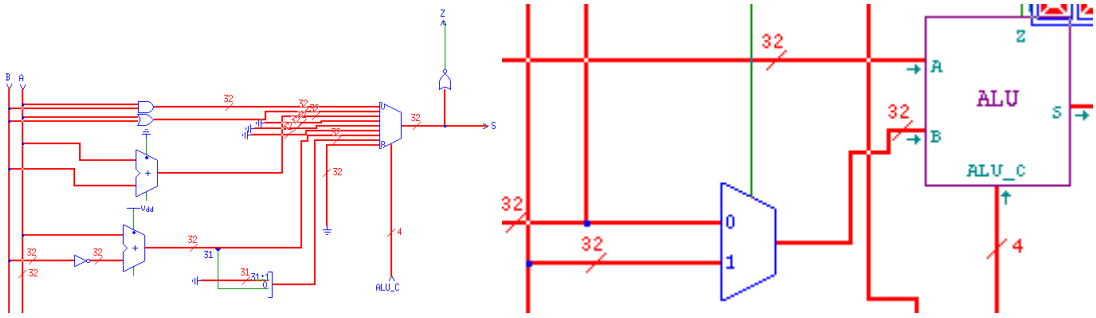


TAREA 3. Utilizando el banco de registros que se proporciona a través del Campus Virtual, realizad la parte del circuito del procesador que se encarga de la ejecución de las instrucciones de aritméticológicas (Formato R) y que se muestra en la siguiente figura.

En esta fase, se describen cómo funcionan la Memoria y la ALU. La ALU (unidad aritmético-lógica) realiza operaciones aritméticas y lógicas en los operandos que recibe, dependiendo de los 4 bits de ALUOperation que recibe desde la Unidad de Control. La ALU también tiene una señal de salida Zero que se activa si el resultado de la operación es cero. La Memoria se refiere a la memoria RAM, que se utiliza para leer y escribir datos en la dirección de memoria que se le indica.

Las instrucciones de formato R se utilizan para operaciones aritméticas y lógicas y se componen de dos registros fuente, un registro de destino, el shamt (en instrucciones de desplazamiento), el funct y el código de operación para la Unidad de Control.

De nuevo, debemos realizar pruebas para comprobar que el formato R funciona mediante el uso de DIPs y LEDs. Ahora como nuevo añadiremos la ALU, por tanto, usando la tabla dada en el enunciado tenemos que:

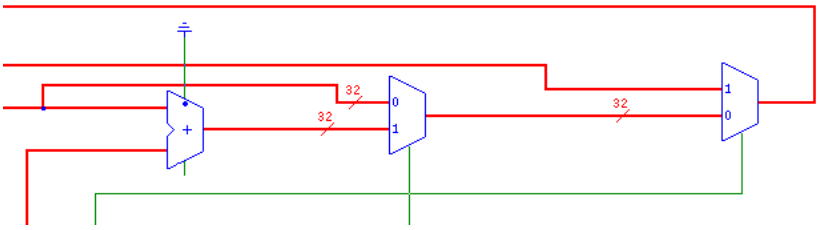


Mediante la señal ALU operation controlaremos que operación sacamos.

TAREA 4. Realizad la parte del circuito del procesador que se encarga de la ejecución de las instrucciones básicas de formato I y J (beq y j) y que se muestra en la siguiente figura

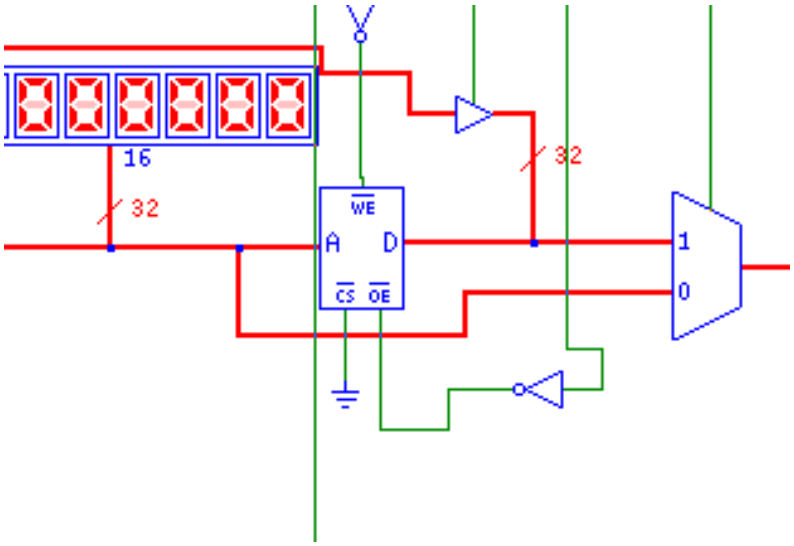
Las instrucciones de formato I se utilizan para cargar o almacenar datos en la memoria y para ramificaciones condicionales. Las instrucciones de formato J se utilizan para saltar directamente a una dirección de memoria. Las instrucciones de formato I incluyen dos operandos para leer o escribir en un registro o en memoria, así como un valor inmediato para definir la instrucción o la dirección de salto. Las instrucciones de formato J incluyen un código de operación para identificar la operación de salto y 26 bits para indicar la dirección de salto, que se forma a partir de los 6 bits más significativos del PC y los 26 bits de memoria.

Aquí comparamos los valores de los registros rs y rt. En la ALU se determina si se va a hacer un salto. Si la comparación es verdad, se calcula la dirección de destino sumando la dirección del PC con el valor de inmediato de la instrucción. Si la comparación es falsa, se continúa ejecutando las instrucciones en secuencia. Para comprobarlo hemos hecho lo mismo de antes: usando DIPs, comprobamos que hace el salto correctamente.



TAREA 5. A partir del diseño de la tarea anterior, añadir la parte del circuito del procesador que se encarga de la ejecución de las instrucciones de acceso a memoria, lw y sw (Formato I) y que se muestra en la siguiente figura.

En esta tarea, añadimos las instrucciones load y store (lw y sw) de formato I. Las señales de control MemRead y MemWrite se encargan de activar y desactivar la lectura/escritura en la memoria de datos. La señal de control MemtoReg selecciona el dato que se escribirá en el banco de registros y la señal de control RegDst selecciona el registro de escritura del banco de registros. Obviamente y como siempre, comprobamos mediante DIPs y LEDs el correcto funcionamiento de la implementación añadida.

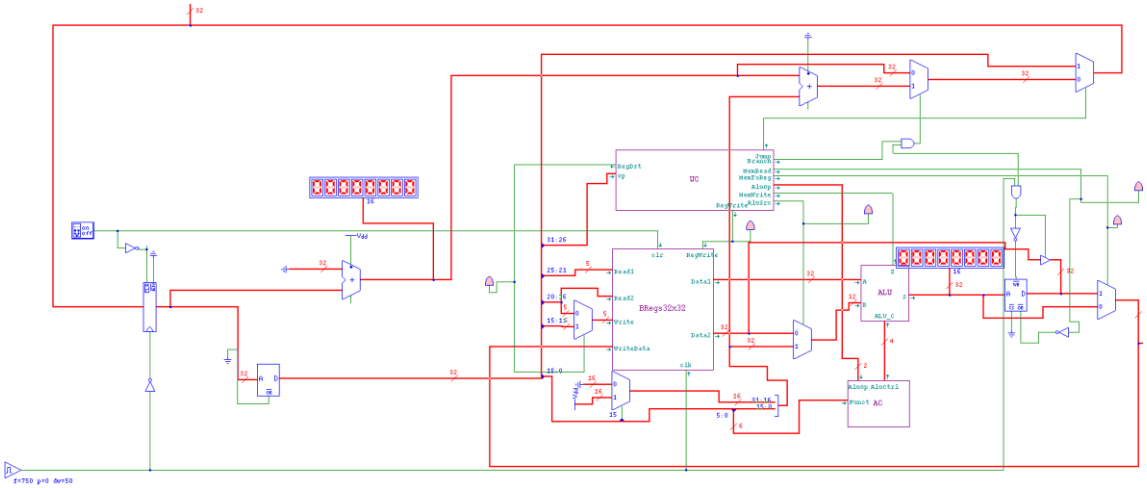


TAREA 6. Unid todos los componentes implementados hasta el momento para obtener la Unidad de Proceso completa del procesador MIPS monociclo. En la siguiente figura se muestra el esquema simplificado

Como bien ya hemos ido haciendo todo en el mismo programa, tenemos todo unido de forma que solo nos quedaría hacer la Unidad de Control en la siguiente tarea.

TAREA 7. Realizad la parte del circuito que se encarga de la Unidad de Control del procesador MIPS y que se muestra junto a la Unidad de Proceso en la siguiente figura.

En esta tarea nos encargamos de hacer la Unidad de Control. Este, se encarga de generar las señales de control. Estas señales se generan gracias a la tabla que nos ofrece el enunciado. A partir del código de operación y función, mediante la interpretación de la tabla y el uso de Karnaughs para simplificar la Unidad de Control, obtenemos las señales de control necesitadas para el correcto funcionamiento del procesador.



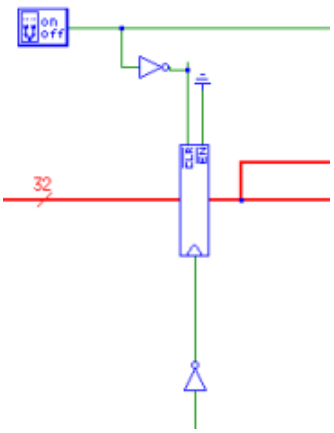
TAREA 10. Realizad una prueba conjunta de funcionamiento de las instrucciones del repertorio ISA básico que se ha asumido en esta práctica, a través de la ejecución del programa de prueba que se proporciona.

Mediante el programa proporcionado y el uso de un archivo.mem y el test.gss, podemos comprobar el correcto funcionamiento del procesador. Después de hacer la simulación, comprobamos que hace la multiplicación de forma correcta

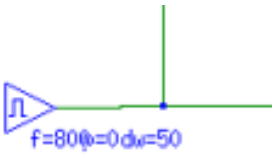
TAREA 11. Realizad un programa ensamblador de prueba propio y evaluad el funcionamiento conjunto de todas las instrucciones del repertorio ISA que se han asumido en esta práctica.

En el juego de pruebas, adjuntaremos el programa usado para comprobar el correcto funcionamiento del procesador.

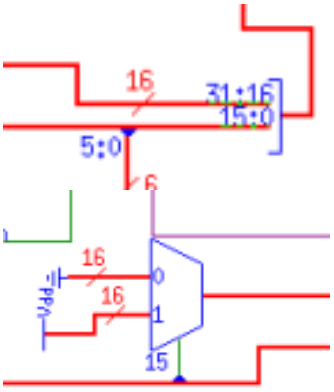
Diseño



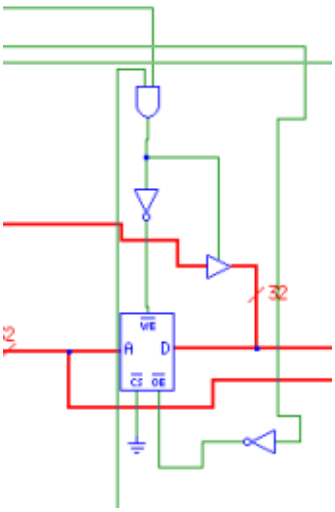
Gracias al uso de las puertas NOT, podemos asegurar el correcto funcionamiento de la lectura de registros y el PC. Ya que cada vez que se detecta el cambio de 0 → 1, hay una actualización del PC gracias al clock, lo que hacemos es, negar el clock y el PC de manera que el flanco de la señal empiece por 1. De esta forma, hacemos que el flanco sea más largo, ya que cuando empieza la señal estamos en 1, y para que haya un cambio de 0 → 1, antes tendrá lugar el cambio de 1 → 0. De esta forma, alargamos la duración de la actualización de apunte de registros (PC) para asegurarnos del correcto funcionamiento de este y evitar problemas en la lectura de otros registros.



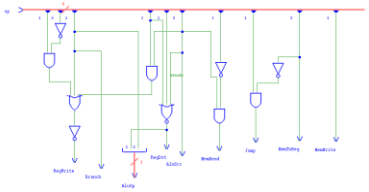
El clock está en 750 para que dé tiempo hacer todo el circuito de ALU y justo el flanco de subida cambie de instrucción en el PC.



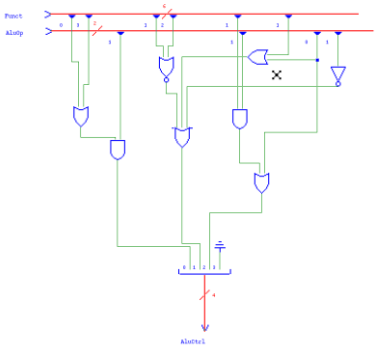
Para hacer el sign extend, aplicamos el wire merge donde gracias al bit 15 miramos si queremos extender f's o 0's. De esta forma podemos expandir los 15 bits sin que haya problemas en el bit de signo.



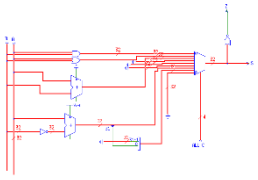
Tenemos una and para sincronizar la señal memWrite y la señal del Clock. Después negamos la señal MemWrite y MemRead porque la memoria usa 0 y no 1. también usamos un buffer triestado para dejar pasar los bits solo si la And es 1.



Hemos aplicado la tabla de verdad y estas son las puertas lógicas que hemos implementado. Usamos los bits de op necesarios.

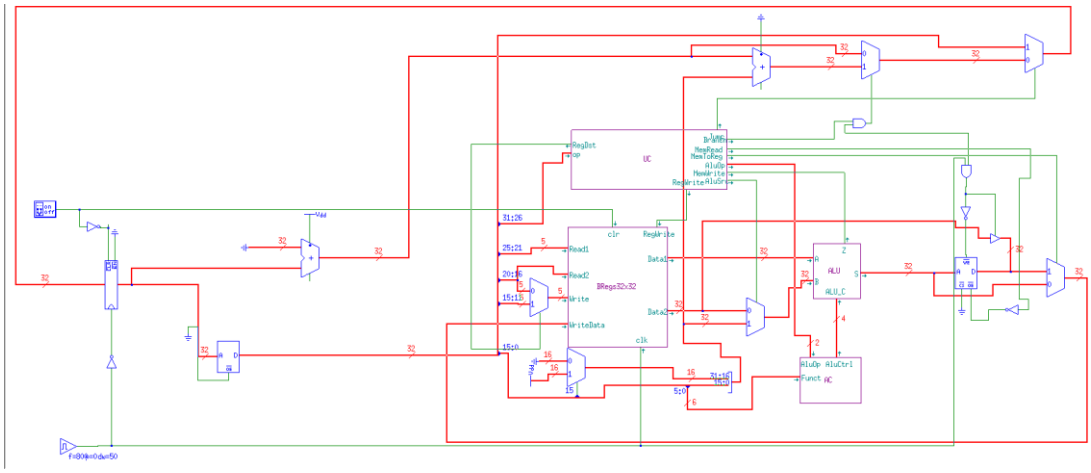


Hemos aplicado la tabla de verdad y estas son las puertas lógicas que hemos implementado. Usamos los bits de Funct y AluOP necesarios.



Hemos decido usar las herramientas del Tkgate, hemos usado las puertas lógicas and y or, el sumador, un sumador, un inversor y vdd para la resta y el bit 31 para la comparación. Con un multiplexor y la señal aluctrl damos el resultado. Finalmente sale también la señal cero si es 0.

Implementación



Juego de pruebas

Nos hemos inspirado en mult.mem. Para comprobar que la resta funciona, hemos implementado un bucle condicional con tres variables. La diferencia es que la primera variable no multiplica si no que la segunda variable resta a 0 tantas veces el valor de la primera variable. Las instrucciones están en residu.mem i ha sido comprobado y verificado con el test2.gss. Este es el resultado:

```
0/ 8c090064 8c040065 8c050066 8024 8824 224402a 11000003 2058022
8/ 2298820 1000fffb ac100067 1000ffff
64/ 1 4 3 fffffff4
```