**Práctica 8:** Procesador RISC

Laboratorio de Organización y Arquitectura de Computadoras

**Grupo:** 2

**Alumnos:**

* Guzmán Sánchez José Emmanuel
* Mejía Ortiz Aarón Enrique
* Sáenz Barragán Ricardo

Objetivo

Diseñar un microprocesador RISC en VHDL, específicamente la versión de pipeline del microprocesador 68HC11 de Motorola.

**Dispositivo**: MAX10 DE-Lite 10M50DAF484C7G

1. Desarrollo

Partimos del diagrama de la arquitectura en pipeline para el microprocesador 68HC11 de Motorola. Se nos proporcionó cada bloque de la arquitectura, por lo que sólo fue necesario implementar las instrucciones que se pedían.

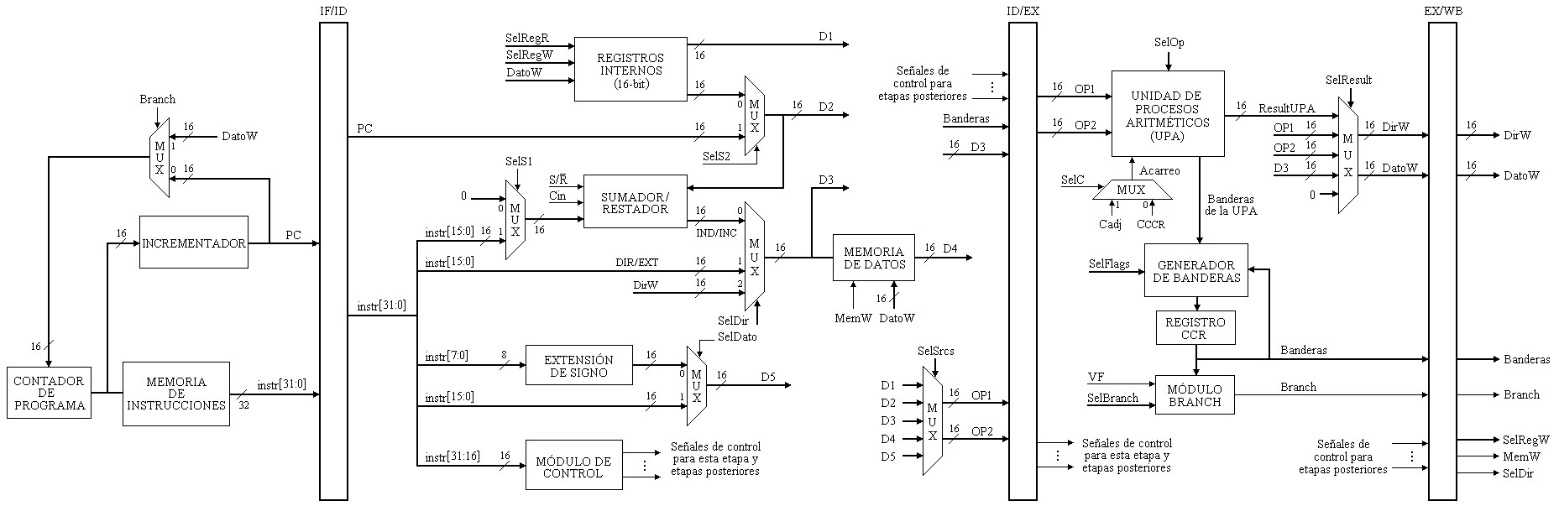


Figura 1. Diagrama del microprocesador 68HC11 de Motorola en su versión pipeline

Primero fue necesario implementar las instrucciones que nos permitieran guardar un dato en los acumuladores con acceso inmediato. Una vez se tenían las instrucciones de carga, se implementó la instrucción *ABA* que suma el valor del acumulador B en el acumulador B. Lo anterior lo repite indefinidamente, por lo que también se implementó la instrucción JMP, que nos permite dar un salta a la instrucción que le indiquemos.

Debido a la dependencia de datos entre la instrucción *ABA* y el contenido del acumulador A, fue necesario implementar la instrucción NOP, para esperar a que se guardará el valor y posteriormente leerlo para volver a realizar la suma.

Debido a que las cargas son por acceso inmediato y las instrucciones de la arquitectura RISC ya contiene los operandos, entonces el contenido de la memoria queda como se muestra en la figura 3.

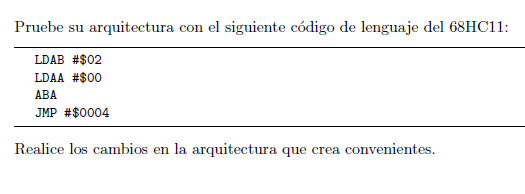
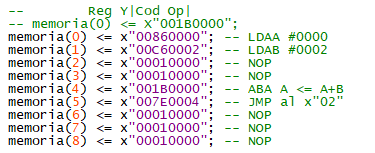
 

Figura 2. Instrucciones a ejecutar Figura 3. Diagrama del registro CCR (de estados y banderas)

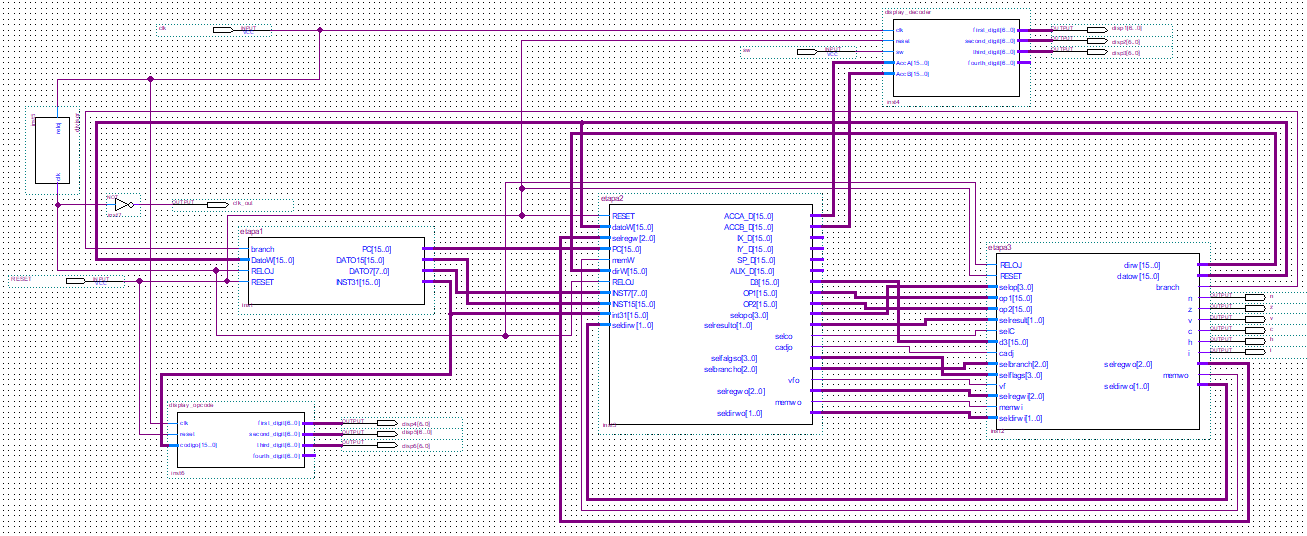


Figura 4. Diagrama de bloques del microprocesador.

Ya con el contenido de la memoria, implementamos las instrucciones. Para esta arquitectura sólo se deben activar las señales necesarias para cada etapa, un proceso totalmente distinto a la arquitectura CISC en donde se tenía que definir la carta ASM y activar las banderas necesarias en cada estado.

La ventaja que se tiene es que desde el principio se activan las banderas para cada una de las etapas, y los registros de acoplamiento se encargan de distribuirlos a las etapas siguientes junto con las señales y datos necesarios.

A pesar de que son instrucciones con un formato totalmente distinto, en esencia es lo mismo, ya que es necesario definir los operandos, la operación, el destino, el origen de los datos, el origen de la siguiente instrucción a ejecutar, etc.

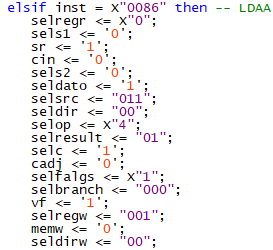
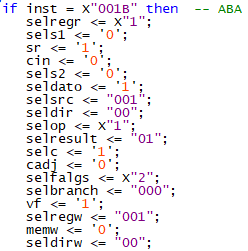


Figura 5. Señales para cada instrcción

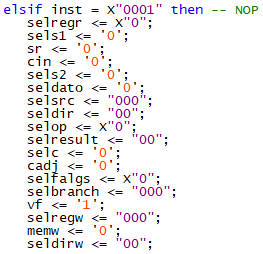
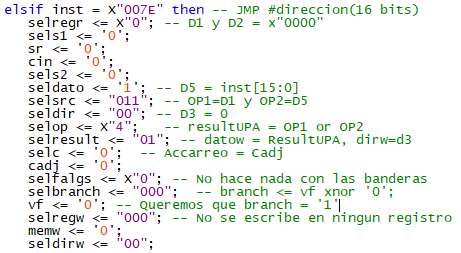


Figura 5. Instrucciones a ejecutar Figura 6. Programa a ejecutar

1. Simulaciones

Las salidas *disp1, dips2 y disp3* corresponden al valor de los acumuladores, los cuales los elegimos por medio de un switch. Las salidas *dis4, disp5 y disp6* corresponden al código de operación de cada instrucción que se ejecuta. Estas salidas están codificadas en lógica negada para display de 7 segementos.

Podemos ver que el acumulador B guarda el valor *2* después de 5 ciclos, ya que se ejecuta primero la operación LDAA y después LDAB, por lo que son 4 ciclos para que se complete la operación y un ciclo de desfase en el pipeline por la primera instrucción.

Para el acumulador B, podemos ver que empiezan su valor en 0, sin embargo, a los 4 ciclos se activa la bandera Z, que es cuando cargamos el valor en acumulador. 4 ciclos después se efectúa la operación ABA, son 4 ciclos porque debemos de esperar 2 ciclos a que se cargue el valor en al acumulador B para poderlo utilizar, de lo contrario leeríamos valores erróneos.

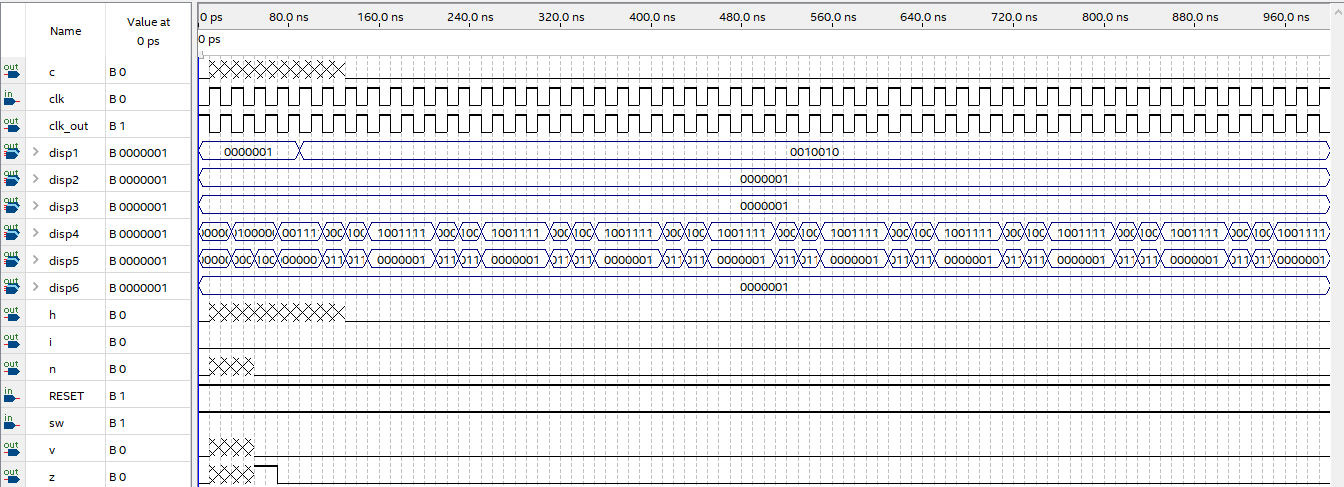


Figura 10. Contenido del acumulador B

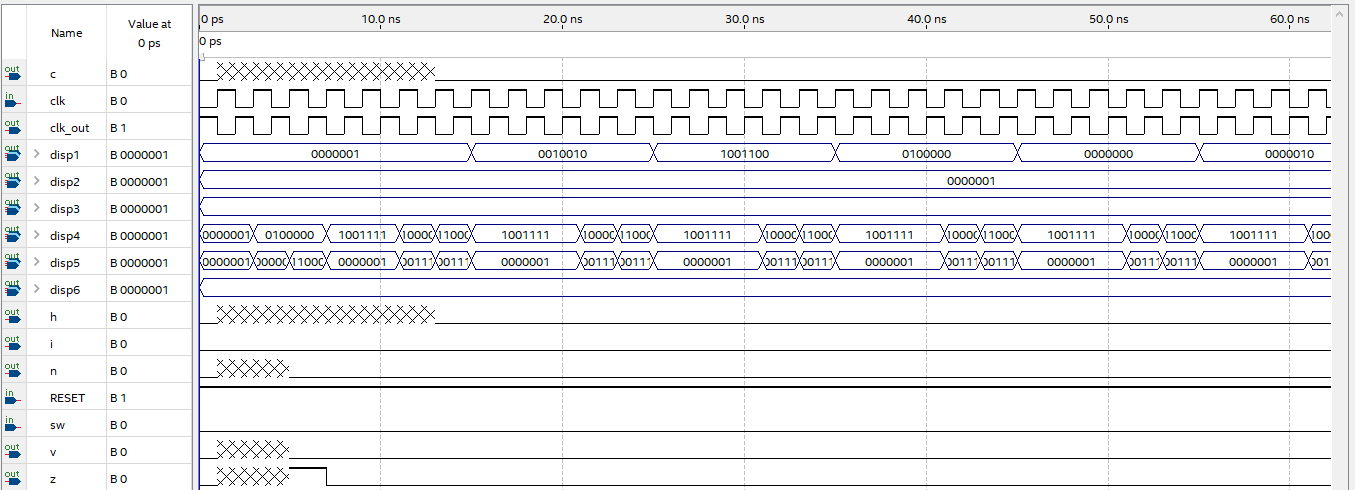


Figura 11. Contenido del acumulador A

1. Conclusiones

Guzmán Sánchez José Emmanuel

Para esta práctica fue fundamental comprender la dependencia de datos, ya que las instrucciones que implementamos requerían que las datos ya estuvieran cargados en los acumuladores, por lo que era necesarios esperar algunos ciclos, con ayuda de la instrucción NOP, para que se guardaran los valores y posteriormente poderlos leer. En comparación con la arquitectura CISC, es mucho más sencilla ya que evitamos acceder a la memoria por los datos al agregarlos directamente en la instrucción. Es mucho más eficiente, sin embargo es necesario tener cuidado con la dependencia de datos.

Mejia Ortiz Aarón Enrique

.

Sáenz Barragán Ricardo

.