Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías

División de Electrónica y Computación
Departamento de Ciencias Computacionales
Licenciatura en Ingeniería en Computación
Arquitectura de computadoras
Clave: CC210 Sección: D02
19:00 – 20:55 Martes Jueves
Actividad 10 “Data Path”
Berrospe Barajas Héctor Eduardo
16-Mayo-2016
López Arce Delgado Jorge Ernesto

Objetivo

El alumno conocerá la estructura de los componentes básicos internos de un procesador.

Introducción

En esta actividad se realizara un Datapath, construiremos los elementos más básicos para ejecutar algunas de las instrucciones de MIPS.

Elementos:

- CP (contador de programa)
- Memoria ROM con las instrucciones precargadas
- Memoria RAM
- Banco de registros
- ALU
- Multiplexores
- Extensión de señal
- Unidad de control de la ALU
- Unidad de control principal

Procedimiento

Explicaremos el funcionamiento de cada uno de los elementos que necesitaremos para realizar el datapath, se representaran de color azul las entradas y color rojo las salidas

CP:

- CP : Entrada de un bit que sirve para indicar que aumente en uno la salida
- Out_addr_ROM : Salida de 4 bits, inicializa en 1111, cada que cambia la entrada CP aumenta en 1, esta salida es la entrada de la ROM

El contador de programa podríamos decir que es un auxiliar, puesto que la función que realiza es aumentar en uno la dirección de memoria de la ROM contenedora de las instrucciones, para de esa manera leer secuencialmente las instrucciones precargadas en la ROM. Cada que la unidad de control manda una señal de que ya termino la primera instrucción el contador de programa realiza su función.

ROM_I

- Addr_ROM: Entrada que recibe del CP, indica la dirección de memoria que va mostrar la salida “d”
- D: Muestra el contenido de la memoria en la dirección especificada

Cada que el contador de programa cambia la dirección de memoria la ROM emite la salida de los datos que se encuentran en la dirección de memoria mandado por el contador del programa.
La ROM en esta práctica contiene puras instrucciones ya sean R o I

RAM

- W_en: Habilita la escritura en la RAM
- R_en: Habilita la lectura en la RAM
- Clk: Cada flanco de subida lee o escriba, según la señal de habilitación que está encendida, en la dirección de memoria que contenga la señal Addr.
- Addr: Especifica la dirección de memoria de la RAM
- Dato: Son los bytes que se escribirán en la memoria RAM
- resRAM: En caso de que este leyendo, la salida mostrara lo que lee en la dirección especificada.

La RAM con un espacio en memoria de 16x32 , cada que existe un flanco de subida en la señal de reloj verifica cuál de las señales de habilitación está encendida, según sea el caso escribe en la posición de memoria que le indican o muestra la posición de memoria en la que e indican.

Banco de registros

- RegWrite: Es el habilitador de escritura en el banco de registros, si su valor es 1 es permitido escribir en algún registro
- Clk: La señal de reloj controla los registros dentro del banco de registro
- Reset: Elimina los datos que se encuentren en los registros
- RR1: Read Register, lee el registro que se encuentra en la posición del valor recibido
- RR2: Read Register, lee el registro que se encuentra en la posición del valor recibido
- WR: Write Register, escribe el registro que se encuentra en la posición del valor recibido
- DW: Son los datos a escribir
- DR1: Devuelve la salida leída de RR1
- DR2: Devuelve la salida leída de RR2

El banco de registros, como lo habíamos realizado previamente en una de las actividades está formado por muchos registros, y recordemos que un registro es un flip flop pero con más capacidad, por lo tanto el funcionamiento no es muy diferente al de la práctica anterior, la diferencia es que en este banco de registros debe de tener la posibilidad de leer 2 diferentes posiciones en memoria y sacar el dato de los registros seleccionados, también es posible guardar datos y resetear los registros para que no contengan nada.

ALU

- Op1: Operando 1
- Op2: Operando 2
- Sel_op: Selecciona la operación que se va a efectuar
- Res: Devuelve el resultado de la operación efectuada sobre el operando 1 y 2
- ZF: Zero flag

Sel_op	Operación
0000	AND
0001	OR
0010	ADD
0110	SUBTRACT
1100	NOR

Zero flag es 1 cuando el resultante es 0, de lo contrario es 0.

Multiplexor

- Sel: Selecciona una de las 2 entradas y permite la salida del seleccionado
- A: Dato
- B: Dato
- M: Dato seleccionado

El multiplexor selecciona una de varias entradas y emite la salida de solo una de las entradas.

Extensión de señal

- Extend: Dato 16 bits
- Extended: Transforma el dato de 16 bits a 32

La extensión de señal, su funcionamiento es bastante sencillo, para hacer un ejemplo practico diremos lo siguiente:

0000000000000000001 = 1 bit
001 = 1 bit

Por lo tanto la función de este módulo es que teniendo un valor en binario dado de 16 bits lo transforme en 32 bits, su comportamiento no es para nada difícil, puesto que tenemos un bus de 16, lo único que se hace para transformarlo es copiar el ultimo bit del bus 16 veces más en nuevo bus, en este caso el bit más significativo es el que se repite hasta llegar a los 32 bits

ALU_unidad de control

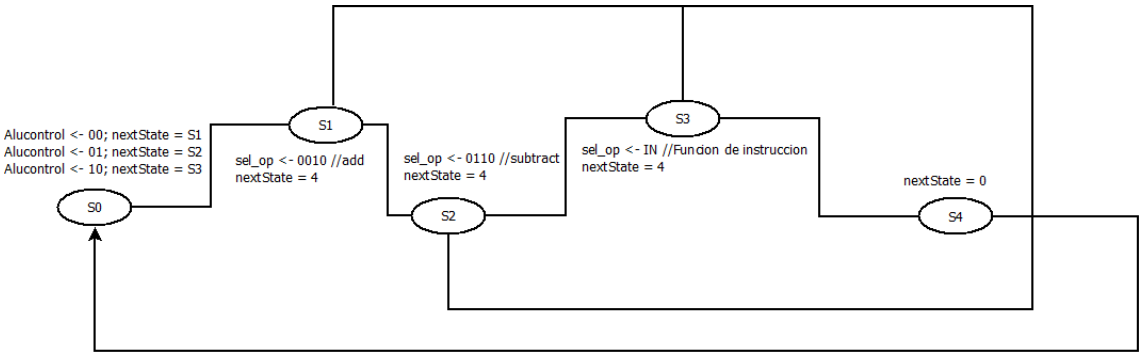
- **Clk:** Señal de reloj, cambia de estado cada flanco de subida
- **ALUcontrol:** Señal de 2 bits que controla la ALU
- **IN:** Entrada de datos recibida por la instrucción, esta representa la operación que realizara la ALU
- **Sel_op:** Es la señal emitida a la ALU, la cual indica la operación de la ALU

Las unidades de control en este caso son las más laboriosas, pues fueron diseñadas con una máquina de estados de tipo **Moore,** puesto que las salidas son válidas después de un flanco de reloj.

Comportamiento según ALUcontrol

ALUcontrol	Realiza
00	ALU add
01	ALU subtract
10	Realiza las operaciones que indiquen la función de la instrucción, en este caso se encuentran en la entrada IN, que son los últimos 5 bits de la instrucción

La máquina de estados es la siguiente:

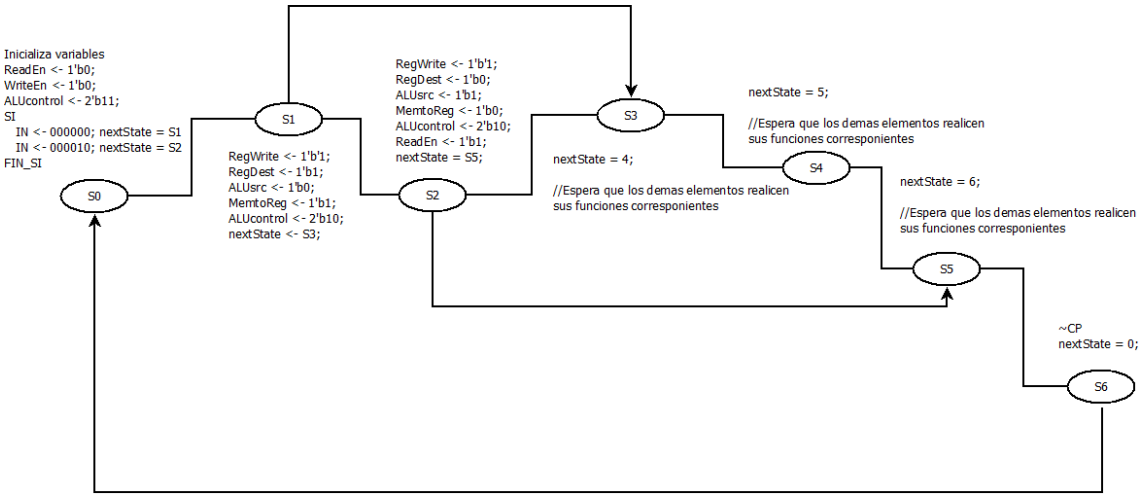


Unidad de control principal

- **Clk:** Señal de reloj, cambia de estado cada flanco de subida
- **IN:** Señal de 6 bits, indica que tipo de instrucción es, R o I
- **RegWrite:** Habilita la señal de escritura del banco de registro
- **RegDest:** Esta señal va conectada a uno de los multiplexores para decidir que dato va dejar pasar
- **CP:** Esta señal cambia cada que se termina un ciclo de la máquina de estados
- **ALUcontrol:** Dependiendo de la instrucción que es, decide que debe de hacer la unidad de control de la ALU
- **ALUsrc:** Esta señal va conectada a uno de los multiplexores para decidir que dato va dejar pasar
- **ReadEn:** Habilita la lectura en la RAM
- **WriteEn:** Habilita la escritura en la RAM
- **MemtoReg:** Esta señal va conectada a uno de los multiplexores para decidir que dato va dejar pasar

La máquina de estados empleada en la unidad de control principal también es del tipo **Moore,** puesto que las salidas son válidas después de un flanco de reloj.

La máquina de estados es la siguiente:

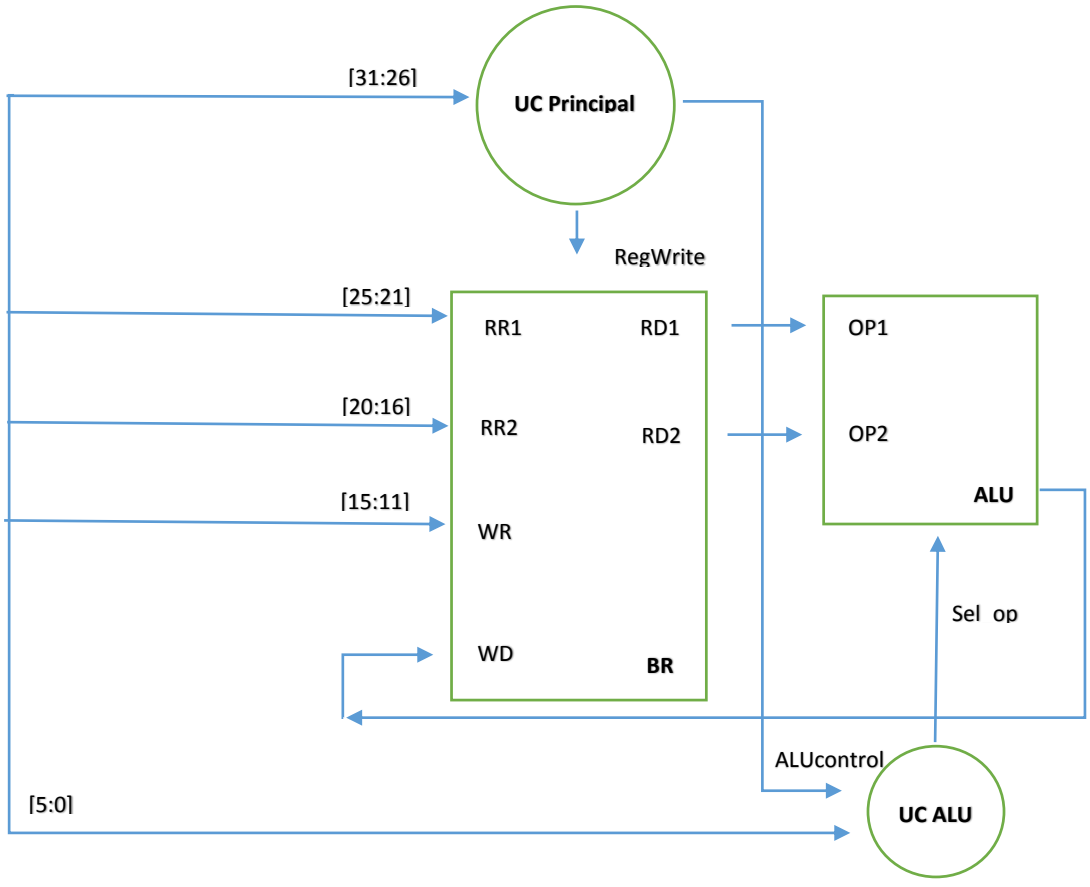


Para entender las máquinas de estados y el por qué existe cada uno de los elementos que realizaremos para nuestro datapath, explicaremos como se ejecuta una instrucción tipo R y una instrucción tipo I.

Instrucción tipo R

1. UC Principal identifica que tipo de instrucción es, este caso sabemos que realizara la R, habilita la señal RegWrite y manda la señal correspondiente a la UC ALU.
2. El banco de registros envía a la ALU los datos que se encuentran en los registros que se introdujo y determina la dirección del registro en donde vamos a escribir.
3. La UC ALU envía el dato para que la ALU determine qué operación realizara.
4. La ALU finalmente realiza la operación y manda el resultado al banco de registros, de esa manera el banco de registros escribe el resultado en la dirección indicada.

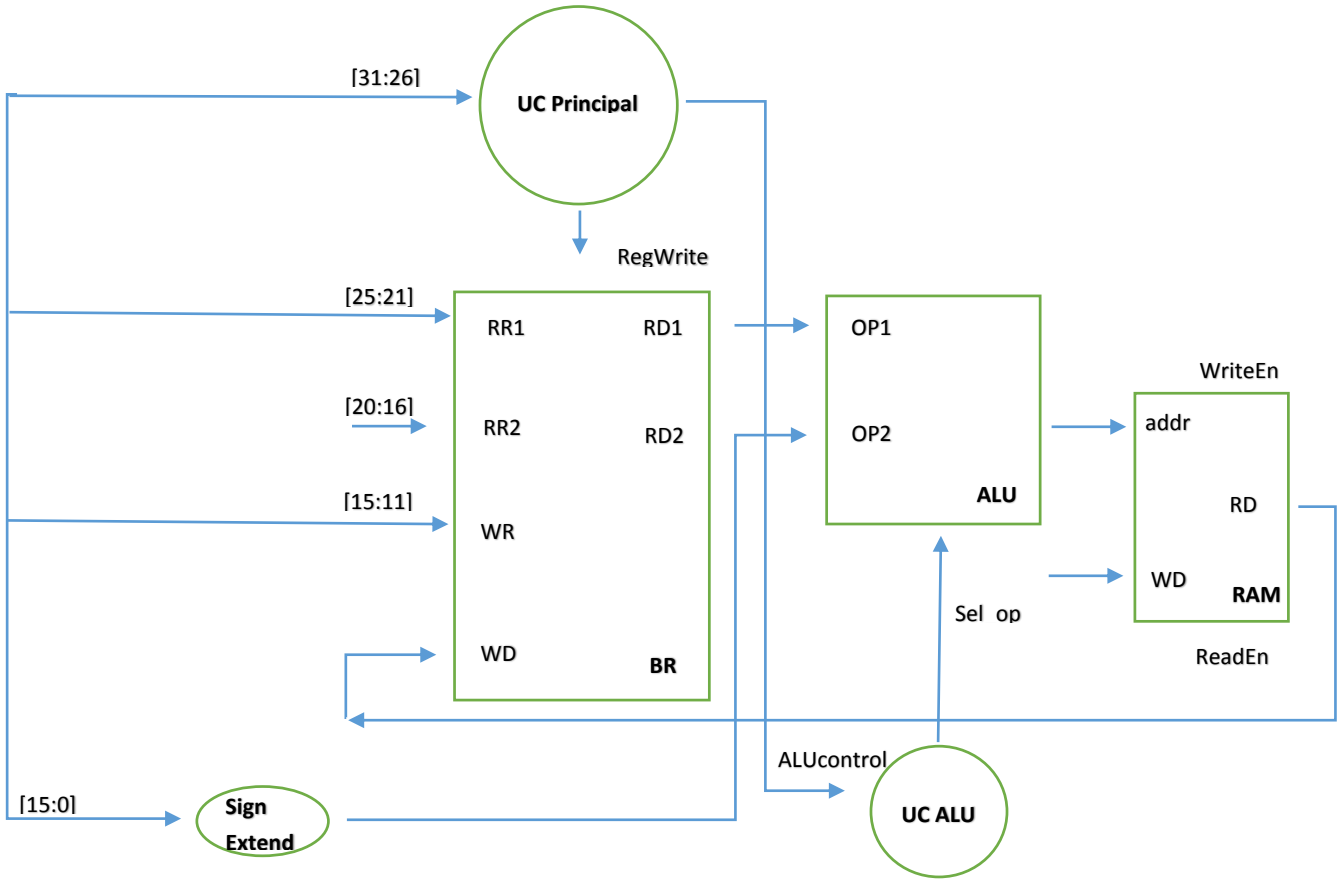
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						RS					RT					RD					SHAMT					FUNC					



Instrucción tipo I

1. UC Principal identifica que tipo de instrucción es, este caso sabemos que realizara la I, habilita la señal RegWrite y manda la señal correspondiente a la UC ALU.
2. El banco de registros envía a la ALU los datos que se encuentran en el registro que se introdujo y determina la dirección del registro en donde vamos a escribir.
3. La UC ALU envía el dato para que la ALU determine qué operación realizara.
4. La ALU finalmente realiza la operación y manda el resultado a la RAM, este resultado será la dirección de la RAM, la RAM enviara los datos que se encuentren en esa posición hacia el banco de registros para que el banco de registros almacene lo que la RAM envió.

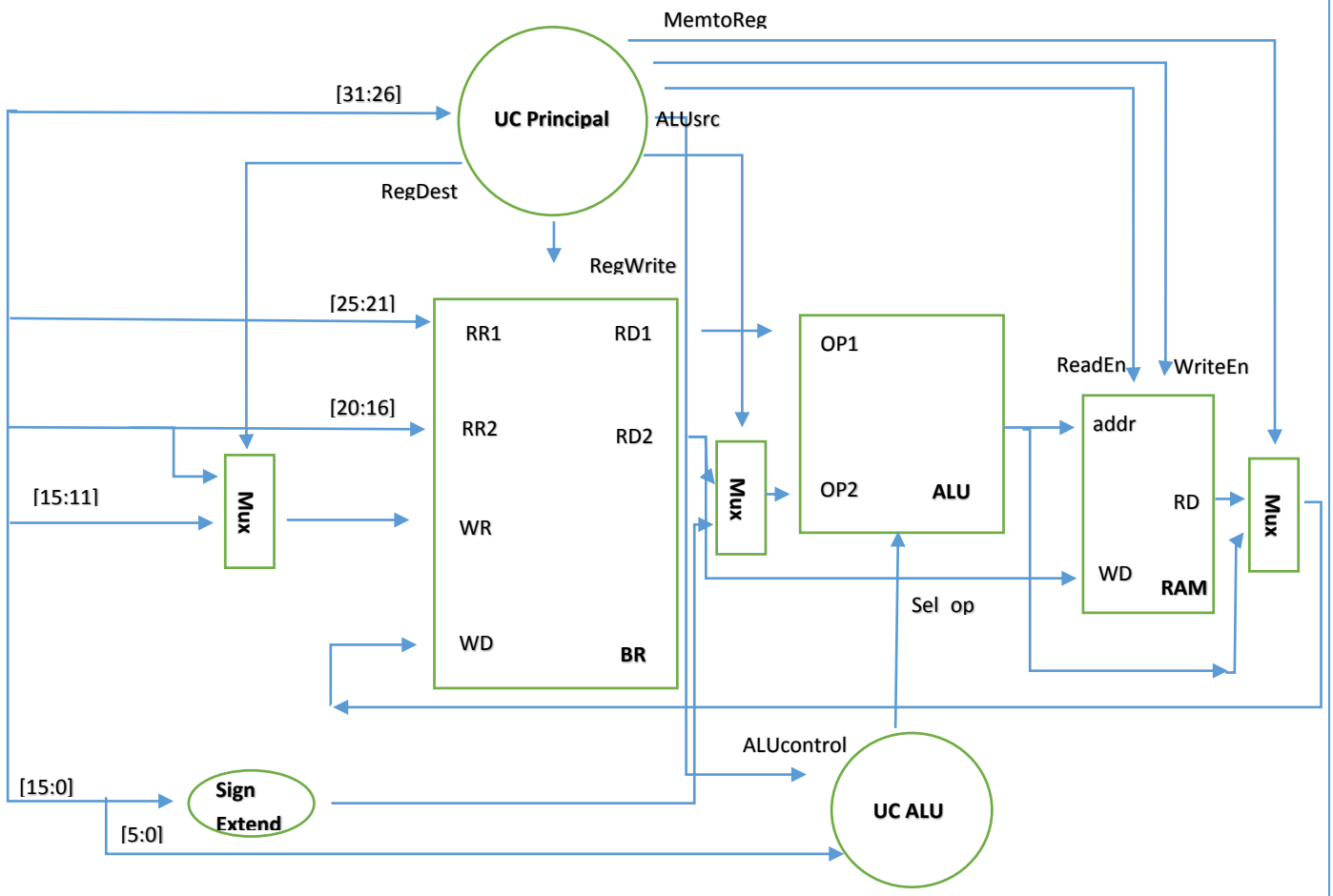
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						RS					RT					16-BIT INMEDIATO																



Nuestra actividad debe de reconocer las dos instrucciones, por lo tanto tenemos que unirlas, una vez entendiendo la ejecución de cada una de las instrucciones será muy sencillo, pues para eso necesitamos los 3 multiplexores que se describían anteriormente, es decir, en caso de que se introduzca una instrucción R o I, se habilitaran las entradas que sean necesarias para ejecutar la instrucción.

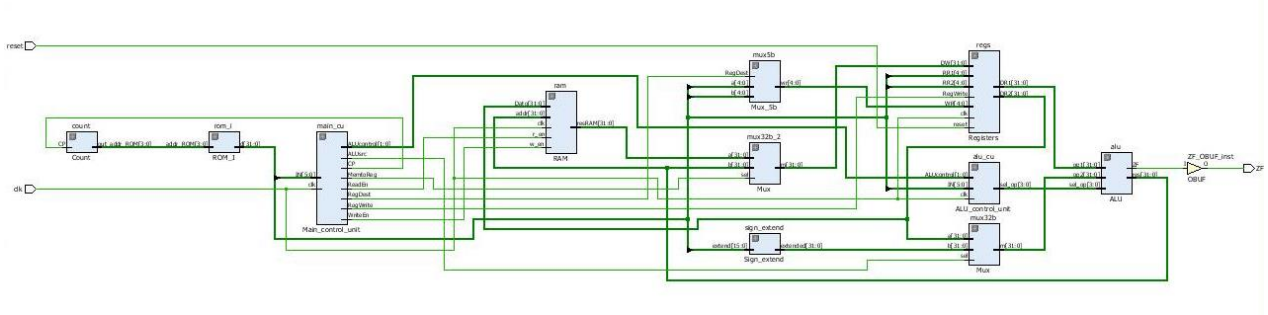
Instrucción tipo R e I.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						RS					RT					RD					SHAMT					FUNC					

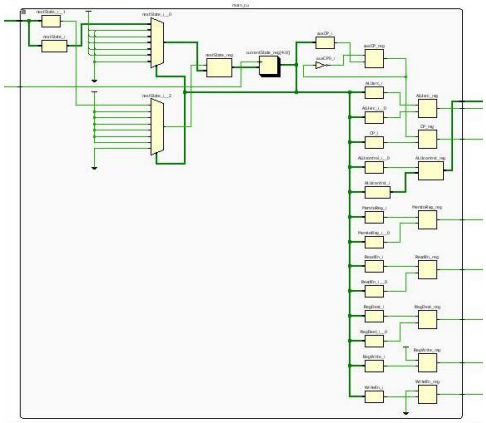


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						RS					RT					16-BIT INMEDIATO																

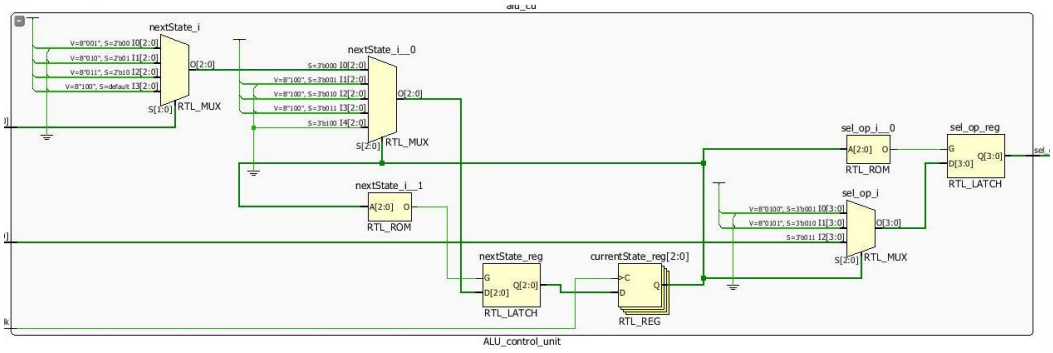
Producto final en verilog, RTL



Unidad de control principal



Unidad de control ALU



Conclusiones

En esta actividad realizamos el datapath, la mejor manera de realizar este tipo de programas es realizar y probar cada módulo individualmente, seguido de eso ensamblar lo necesario para la instrucción R, entenderla bien y hacer una prueba, ya que salga todo bien, realizamos la instrucción I de la misma manera, realizamos modulo por modulo, ya estando bien los códigos los ensamblamos y combinamos con el tipo R. Es interesante ver como se ejecutan las instrucciones, en estos momentos estoy un poco familiarizado con el lenguaje ensamblador, puesto que realice un ensamblador de 64 bits para la arquitectura HC12 recientemente y tengo mejor idea de cómo funcionan viendo la arquitectura de un procesador MISP.