Objetivo

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

Introducción

Pipeline

El pipeline, es una técnica utilizada en el diseño e implantación de microprocesadores en la cual múltiples instrucciones pueden ejecutarse simultáneamente.

Esto no reduce el tiempo que tarda una instrucción en ejecutarse, sólo incrementa el número de instrucciones que se ejecutan simultáneamente; es decir, mejora el rendimiento incrementando la productividad de las instrucciones en lugar del tiempo de ejecución de las instrucciones individuales. Al igual que en la línea de ensamble, el trabajo que se realiza para cada instrucción se descompone en partes más pequeñas; cada una de estas partes, denominadas etapas o segmentos, necesitan una fracción del tiempo total para completar la instrucción. Entre dos etapas de la línea de ensamble se coloca un registro de acoplo, también denominado registro de segmentación, que se encarga de guardar los datos y las señales de control necesarias para etapas posteriores.

Figura 1 Etapas del pipeline

De la figura 1 podemos observar que la ejecución de una instrucción consta de cuatro etapas: 1) traer la instrucción, 2) decodificarla, 3) traer operandos, y 4) ejecutarla. Si cada uno de estas etapas es independiente de los demás, entonces, en el tiempo 'n' se empezaría a traer la instrucción I_n; al mismo tiempo se estaría decodificando la instrucción I_{n-1}, trayendo los operandos de la instrucción I_{n-2} y terminando de ejecutar la instrucción I_{n-3}. Esto significa que una vez que el cauce está lleno, idealmente, en cada ciclo de reloj se estaría ejecutando una instrucción.

Adicionalmente, debido a que las etapas no están sincronizadas, el tiempo por instrucción en una arquitectura segmentada no tiene el valor mínimo posible y la mejora en velocidad será menor que el número de etapas.

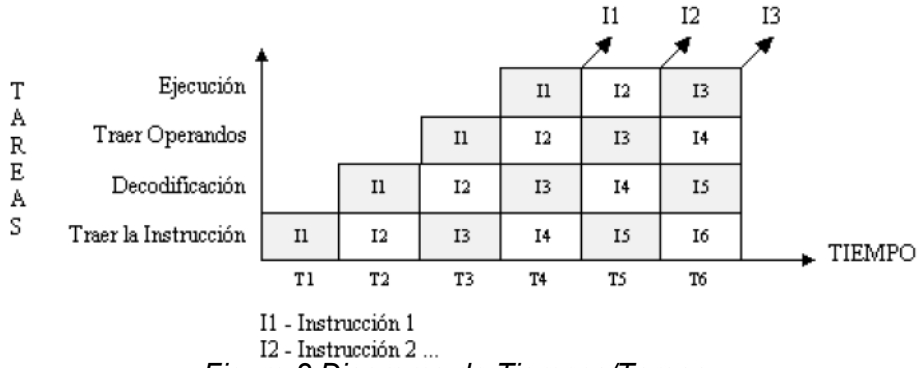

Figura 2 Diagrama de Tiempos/Tareas

Arquitectura segmentada del 68HC11

En el 68HC11 notaremos que cada instrucción ejecuta una serie de pasos. En general, los pasos a seguir son los siguientes:

1. Traer de la memoria la instrucción que se desea ejecutar (ciclo fetch)
2. Decodificación de la instrucción
3. Si la instrucción requiere leer un operando de la memoria, entonces se calcula la dirección efectiva de ese operando y se lee el dato de la memoria
4. Si lo requiere la instrucción, se leen de los registros internos del microprocesador los operandos necesarios
5. Ejecución, es decir, se realiza una operación en la unidad de procesos aritméticos con los operandos leídos anteriormente
6. Se guardan los resultados de la operación y se actualiza el registro de banderas

Observe que estos pasos son similares a los ejecutados en las cartas ASM para las instrucciones vistas en el capítulo VI. La arquitectura segmentada del 68HC11 también ejecutará estos mismos pasos, pero agrupados en las siguientes cuatro etapas.

1. Etapa IF (instruction fetch). La instrucción a ejecutar es leída de la memoria de instrucciones
2. Etapa ID (instruction decode). Se decodifica la instrucción y se traen los operandos necesarios por la instrucción (tanto de memoria como de registros internos)
3. Etapa EX (execution). Se procesan los operandos en la UPA (unidad de procesos aritméticos)
4. Etapa WB (write back). Se guardan resultados

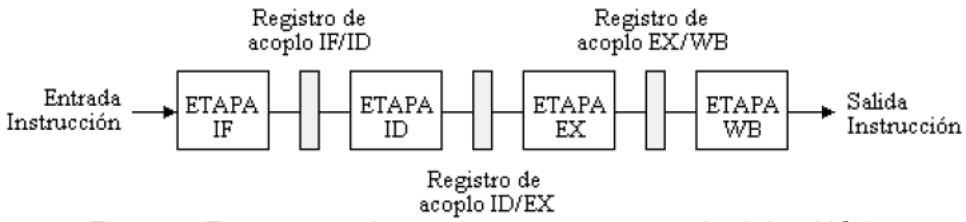

Figura 3 Etapas para la arquitectura segmentada del 68HC11

- *Etapa 01 – Lectura de la instrucción*

Una manera de evitar los accesos a memoria repetidamente es leyendo en una sola pasada toda la información que la instrucción vaya a necesitar, esto es, leer el código de operación de la instrucción, leer los datos inmediatos y leer las direcciones de memoria. Para ello, el tamaño de la instrucción para el 68HC11 segmentado ha sido extendido a 32 bits, los cuales contendrán toda la información necesaria según el modo de direccionamiento del que se trate. Además, la memoria externa será separada en memoria de instrucciones o programa y en memoria de datos.

En esta etapa se lee de la memoria la instrucción a ejecutar y se almacena en el registro de segmentación IF/ID. La dirección de esta instrucción está dada por el contador de programa (PC); dicha dirección se incrementa y se vuelve a cargar en el PC para ser leída en el siguiente ciclo de reloj.

Finalmente, el hardware para la etapa de la lectura de la instrucción queda de la siguiente manera

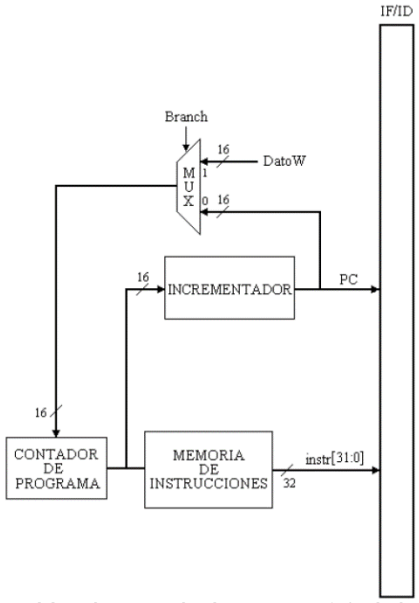

Figura 4 Hardware de la etapa 01 del pipeline

- *Etapa 02 – Decodificación de la instrucción, dirección efectiva y lectura de operandos*

Durante esta etapa, la instrucción leída en la etapa anterior es decodificada. Una vez que se

conoce la instrucción a ejecutar, el módulo de control genera las señales de control que gobernarán el hardware de esta etapa y de etapas posteriores. El hardware para esta etapa se muestra a continuación.

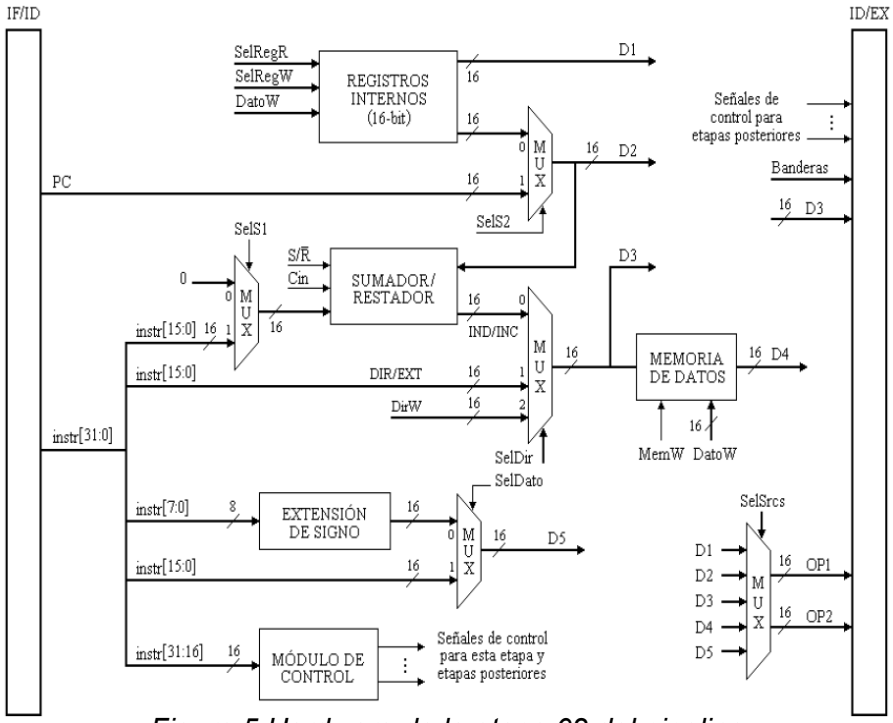

Figura 5 Hardware de la etapa 02 del pipeline

El módulo de control, es el encargado de generar todas las señales de control. Algunas de estas señales serán utilizadas durante esta etapa y otras serán guardadas en los registros de segmentación para su utilización en etapas posteriores.

La generación de las señales de control es posible gracias a la información que le brindan al módulo las líneas instr[31:16], pues estas líneas transportan el código de operación de la instrucción con la información suficiente para saber de qué instrucción se trata y su modo de direccionamiento

Por otra parte, en el módulo de registros internos se encuentran los acumuladores A y B, y los registros internos IX, IY, SP y AUX.

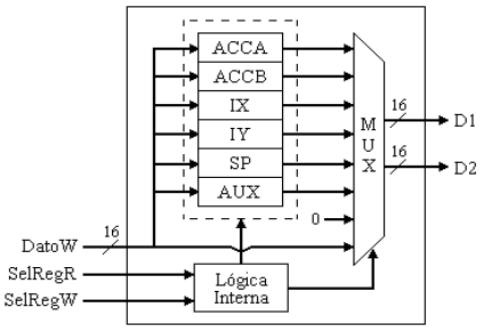

Figura 5.1 Módulo de registros internos.

La línea de control SelRegR le indica al módulo qué registros deseamos leer; la línea SelRegW le indica en qué registro se desea escribir el valor de DatoW; y las salidas D1 y D2 transportan los valores leídos de dichos registros

SelRegW	*Registro que se escribe*
0	Ninguno
1	ACCA
2	IX
3	IY
4	ACCB
5	AUX
6	SP

SelRegR	*Registro seleccionado para lectura*	
	D1	D2
0	0	0
1	ACCA	ACCB
2	ACCB	IX
3	ACCB	IY
4	ACCA	0
5	ACCB	0
6	ACCA	IX
7	ACCA	IY
8	AUX	0
9	0	IX
A	0	IY
B	0	SP
C	ACCA	SP
D	ACCB	SP
E	IX	SP
F	IY	SP

Figura 5.2 Selección registros para escritura.

Figura 5.3 Selección registros para lectura.

Finalmente, existe un multiplexor que selecciona los operandos para la siguiente etapa. Este multiplexor elige estos operandos de los datos presentes en los buses D1, D2, D3, D4 y D5, según la instrucción que se trate.

SelSrcs	*Operandos seleccionados*	
	OP1	OP2
0	0	0
1	D1	D2
2	D1	D4
3	D1	D5
4	D4	D3
5	D2	D5
6	D2	D4

Figura 5.4 Selección operandos para etapa de ejecución

- *Etapa 03 – Ejecución y calculo de banderas y saltos*

En esta etapa se ejecutan tres tareas: operar los operandos obtenidos en la etapa de decodificación (etapa 2); actualizar el registro de estados o banderas; y calcula la condición de salto.

El hardware para esta etapa se muestra a continuación:

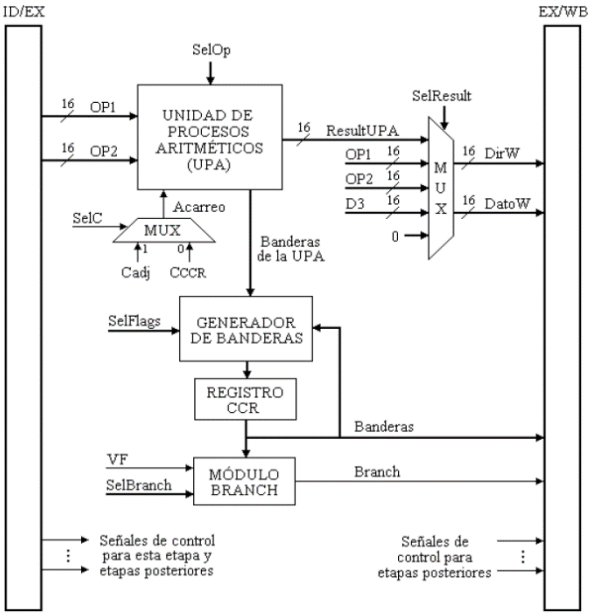

Figura 6 Hardware de la etapa 03 del pipeline

La unidad de procesos aritméticos (UPA) se encarga de obtener el resultado entre los operandos según la operación establecida en SelOp. SelOp es una señal de control generada en la etapa anterior, pero que es empleada hasta esta etapa. En la UPA se realizan las operaciones lógicas, las operaciones aritméticas y los corrimientos.

SelOp	*Operación ejecutada*
0	Ninguna
1	OP1 + OP2 + Acarreo
2	OP1 – OP2 – $\overline{\text{Acarreo}}$
3	OP1 and OP2
4	OP1 or OP2
5	OP1 xor OP2
6	Corrimiento a la izquierda de OP1 con $B0 = 0$
7	Corrimiento a la derecha de OP1 con $B15 = B15$
8	OP2 – OP1 – $\overline{\text{Acarreo}}$
9	Corrimiento a la derecha de OP1 con $B15 = 0$
A	Rotación a la izquierda de OP1 con $B0 = C_{CCR}$
B	Rotación a la derecha de OP1 con $B15 = C_{CCR}$

Figura 6.1 Operaciones de la UPA

El acarreo de entrada al módulo UPA es seleccionado mediante la señal de control SelC. Si SelC vale cero, el acarreo elegido proviene del registro de estados (CCCR); si SelC vale uno, entonces el acarreo elegido es C_{adj}, el cual es generado por el módulo de control de la etapa 2 y es establecido a un cierto valor según la instrucción que se trate.

SelFlags	*Banderas que son actualizadas en el CCR*
0	No se modifica el CCR
1	N, Z, V=0
2	N, Z, V, C, H
3	N, Z, V, C
4	Z
5	C=0
6	I=0
7	V=0
8	C=1
9	I=1
A	V=1
B	N, Z, V=0, C=1
C	N, Z, V

Figura 6.2 Banderas afectadas en CCR

Las banderas que se modificaron tras la operación ejecutada en la UPA son guardadas en el registro de estados por el módulo generador de banderas.

En caso de ejecutar una instrucción de salto, la UPA calculará la dirección a donde probablemente se deba saltar, y el módulo Branch evaluará la condición de salto para determinar si en verdad se ejecuta el salto o no

SelBranch	*Condición a evaluar*
0	Se compara con cero
1	C
2	Z
3	$N \oplus V$
4	$Z + (N \oplus V)$
5	C + Z
6	N
7	V

Figura 6.3 Condiciones de salto, Brach

Por último, son guardados en el registro de segmentación EX/WB el resultado de la UPA, la dirección efectiva obtenida en la etapa 2, algunos valores de banderas, y las señales de control necesarias para la última etapa.

SelResult	*Fuentes seleccionadas*	
	DatoW	DirW
0	0	0
1	ResultUPA	D3
3	OP1	D3

Figura 6.4 Fuentes para la señal de control SelResult.

- *Etapa 04 – Post-escritura*

En las arquitecturas segmentadas las instrucciones y los datos se desplazan generalmente de izquierda a derecha a través de las etapas, sin embargo, hay dos excepciones que se presentan en la etapa de post-escritura:

1. Cuando se guarda el resultado de la UPA en los registros o en memoria, haciendo que se retroceda a la etapa 2.
2. Cuando se selecciona el nuevo valor de PC en la etapa 1, valor que puede ser el PC incrementado, o bien, la dirección de salto calculada en la etapa 3.

En resumen, la etapa de post-escritura se encarga de actualizar los resultados obtenidos en etapas anteriores. Finalmente obteniendo el hardware completo para el pipeline como se muestra a continuación.

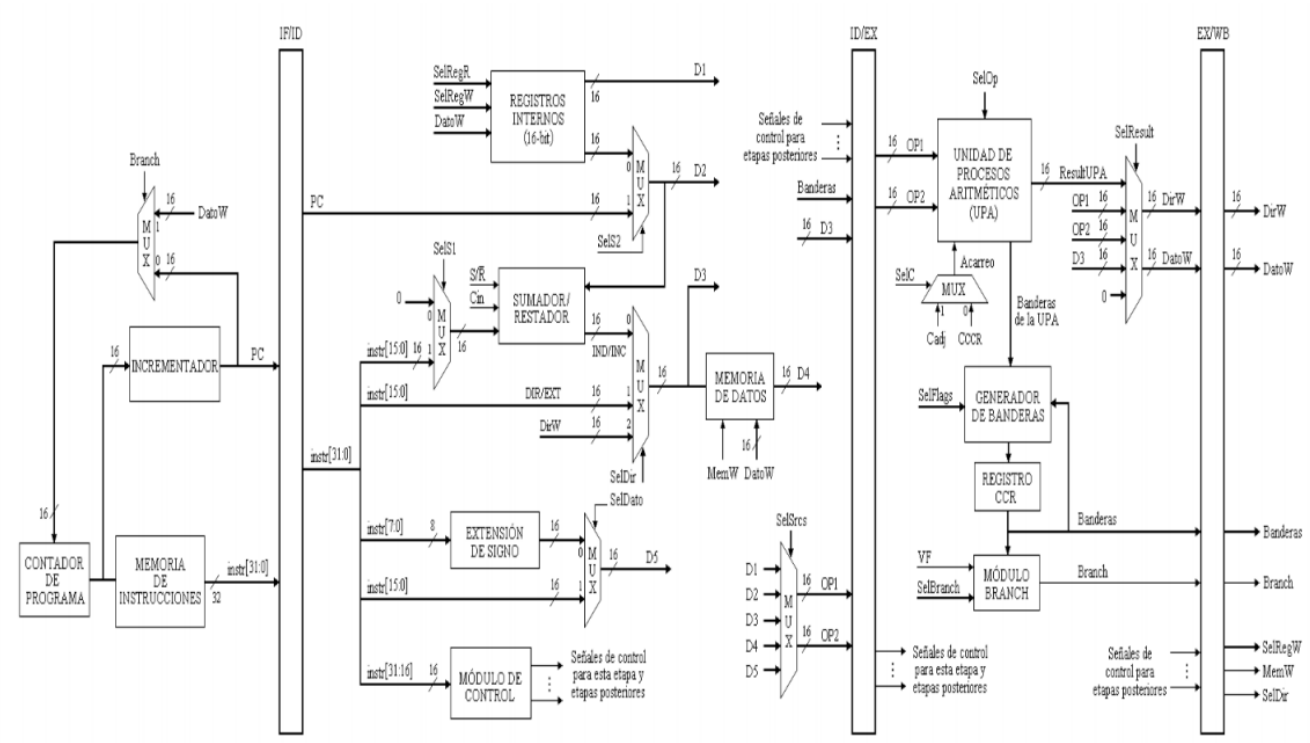

Figura 7 Hardware completo para la arquitectura RISC del 68HC11.

La cual se llevará a cabo en está practica en lenguaje vhdl y se simularan un par de instrucciones para ver su correcto funcionamiento y como se desarrolla.

Desarrollo

Para poder realizar la arquitectura RISC del 68HC11 se nos dio ya construida todo el pipeline que desglosaremos a continuación.

Primeramente, tenemos la etapa 01, que como vimos en la introducción tenemos el contador, el incrementador, el multiplexador y la memoria de instrucciones conectadas entre sí arrojando sus salidas a los registros de PC e instrucciones para ser tomadas posteriormente por la etapa 02

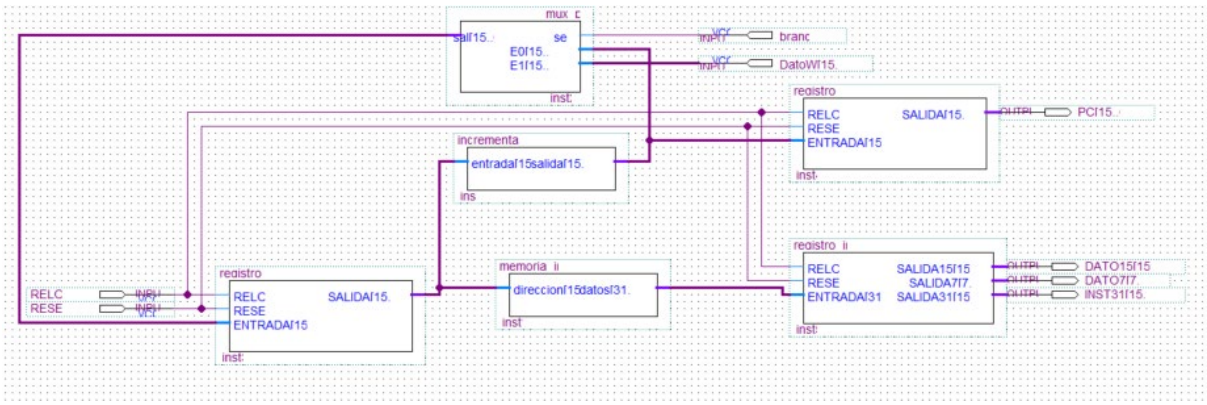

Figura 8 Etapa 01 en vhdl

Posteriormente tenemos la etapa 02 con los bloques que se señalaron en la introducción, destacando los registros internos, el módulo de control, el sumador y el registro de acoplamiento los cuales son fundamentales para poder llevar a cabo el pipeline, teniendo sus respectivas salidas para poderlo conectar a la etapa 03

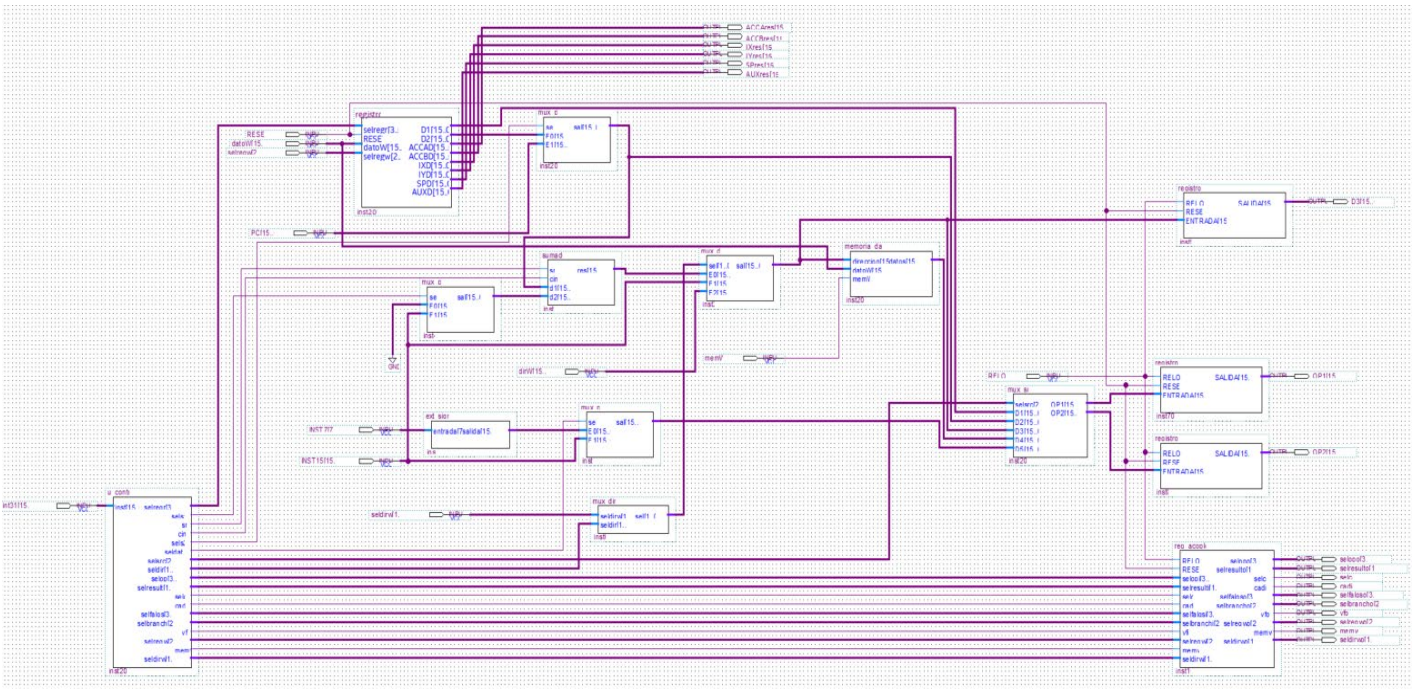

Figura 9 Etapa 02 en vhdl

Finalmente, la etapa 03, ya que la etapa 04 como vimos en la introducción, simplemente son las salidas o las banderas que se activan durante todo el proceso, lo cual se hace de forma directa para poder simularlo de buena forma en vhdl.

Como podemos ver en esta etapa 03 tenemos a la UPA y al generador de bandera, las cuales se muestran de manera independiente, cada una con su bloque para poder controlarlas de mejor manera y pode mostrarlas en la simulación

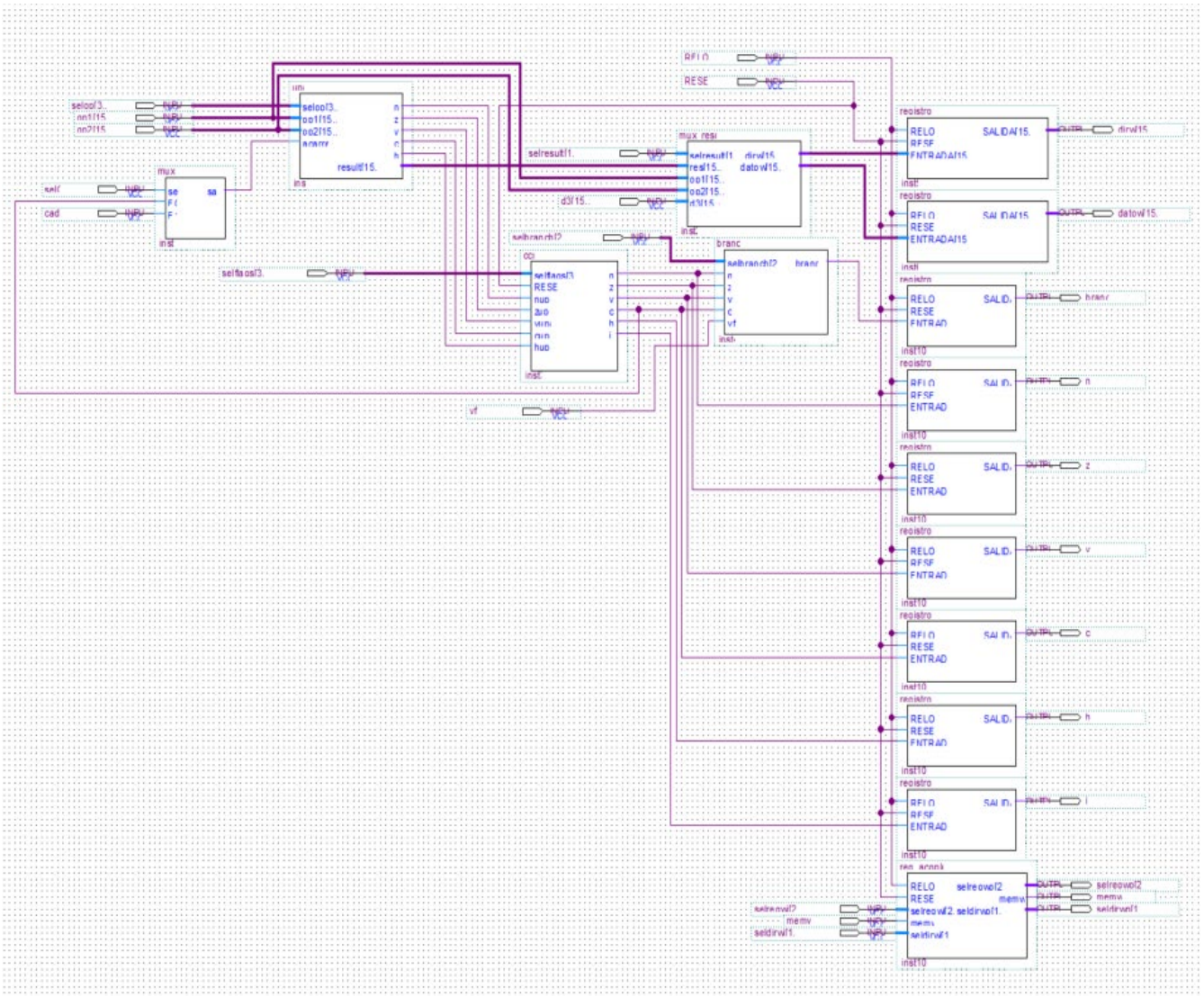

Figura 10 Etapa 03 en vhdl

Ejecución de las instrucciones

Ahora para poder empezar a ejecutar instrucciones necesitamos conocer las señales de control de cada una de ellas para cada una de las etapas que vimos anteriormente.

Para la práctica se solicitaron las siguientes instrucciones:

Código de la práctica
LDAB #0x02
LDAA #0x00
ABA
JMP 0x0004

- *Instrucción NOP*

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	0
	Cin	0
	SelS2	0
	SelDato	0
	SelScrs	0
	SelDir	0
Etapa 3	SelOp	0
	SelResult	0
	SelC	0
	Cadj	0
	SelFlags	0
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

Figura 11 Señales de control para la instrucción NOP

- *Instrucción ABA*

Etapa	Señal de control	Valor
Etapa 2	SelRegR	1
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	0
	SelScrs	1
	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	2
	SelBranch	0
	VF	1
Etapa 4	SelRegW	1
	MemW	0
	SelDirW	0

Figura 11.1 Señales de control para la instrucción ABA

- *instrucción JMP*

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	0
	SelScrs	4
	SelDir	1
Etapa 3	SelOp	0
	SelResult	3
	SelC	1
	Cadj	0
	SelFlags	0
	SelBranch	0
	VF	0
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

Figura 11.2 Señales de control para la instrucción JMP

- *Instrucción LDAA*

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	4
	SelDir	0
Etapa 3	SelOp	5
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	1
	SelBranch	0
	VF	1
Etapa 4	SelRegW	1
	MemW	0
	SelDirW	0

Figura 11.3 Señales de control para la instrucción LDAA

- *Instrucción LDAB*

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	4
	SelDir	0
Etapa 3	SelOp	3
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	1
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

Figura 11.4 Señales de control para la instrucción LDAB

Una vez que tenemos para cada una de las instrucciones las señales de control, las colocamos en la unidad de control:

```
architecture Behavioral of u_control is
begin
	process (inst)
	begin
		case inst is
			when X"0001" => -- NOP
				selregr <= "0000";
				sels1 <= '0';
				sr <= '0';
				cin <= '0';
				sels2 <= '0';
				seldato <= '0';
				selsrc <= "000";
				seldir <= "00";
				selop <= "0000";
				selresult <= "00";
				selc <= '0';
				cadj <= '0';
				selfalgs <= "0000";
				selbranch <= "000";
				vf <= '1';
				selregw <= "000";
				memw <= '0';
				seldirw  <= "00";
			when X"001B" => -- ABA
				selregr <= "0001";
				sels1 <= '0';
				sr <= '1';
				cin <= '0';
				sels2 <= '0';
				seldato <= '0';
				selsrc <= "001";
				seldir <= "00";
				selop <= "0001";
				selresult <= "01";
				selc <= '1';
				cadj <= '0';
				selfalgs <= "0010";
				selbranch <= "000";
				vf <= '1';
				selregw <= "001";
				memw <= '0';
				seldirw  <= "00";
			when X"007E" => -- JMP
				selregr <= "0000";
				sels1 <= '0';
				sr <= '1';
				cin <= '0';
				sels2 <= '0';
				seldato <= '0';
				selsrc <= "100";
				seldir <= "01";
				selop <= "0000";
				selresult <= "10";
				selc <= '1';
				cadj <= '0';
				selfalgs <= "0000";
				selbranch <= "000";
				vf <= '0';
				selregw <= "000";
				memw <= '0';
				seldirw  <= "00";
			when X"0086" => -- LDAA
				selregr <= "0000";
				sels1 <= '0';
				sr <= '1';
				cin <= '0';
				sels2 <= '0';
				seldato <= '1';
				selsrc <= "011";
				seldir <= "00";
```

```
                                selop <= "0100";
                                selresult <= "01";
                                selc <= '1';
                                cadj <= '0';
                                selfalgs <= "0001";
                                selbranch <= "000";
                                vf <= '1';
                                selregw <= "001";
                                memw <= '0';
                                seldirw  <= "00";
                        when X"00C6" => -- LDAB
                                selregr <= "0000";
                                sels1 <= '0';
                                sr <= '1';
                                cin <= '0';
                                sels2 <= '0';
                                seldato <= '1';
                                selsrc <= "011";
                                seldir <= "00";
                                selop <= "0100";
                                selresult <= "01";
                                selc <= '1';
                                cadj <= '0';
                                selfalgs <= "0001";
                                selbranch <= "000";
                                vf <= '1';
                                selregw <= "100";
                                memw <= '0';
                                seldirw  <= "00";
                        WHEN OTHERS => -- Señales por Default
                                selregr <= "0000";
                                sels1 <= '0';
                                sr <= '1';
                                cin <= '0';
                                sels2 <= '0';
                                seldato <= '0';
                                selsrc <= "000";
                                seldir <= "00";
                                selop <= "0000";
                                selresult <= "00";
                                selc <= '1';
                                cadj <= '0';
                                selfalgs <= "0000";
                                selbranch <= "000";
                                vf <= '1';
                                selregw <= "000";
                                memw <= '0';
                                seldirw  <= "00";
                end case;
        end process;
end
```

Figura 12 Código de u_control.vhdl

- *Contenido en memoria del programa*

ID	Código	HH	LL
0	C6	0	2
0	86	0	0
0	1B	0	0
0	7E	0	4

Figura 13 Tabla del contenido de la memoria de para el programa de la práctica

Simulaciones

Para las simulaciones se ocupó el simulador de *Gate Level Simulation* ya que se puede realizar simulaciones con respecto al tiempo y no solo a la funcionalidad.

En donde se muestran las banderas, los acumuladores de A y B (*ACCA y ACCB*), el registro auxiliar (*AUX*), el valor de la PC (*PC*) y los valores del registro X y Y (*IX y IY*)

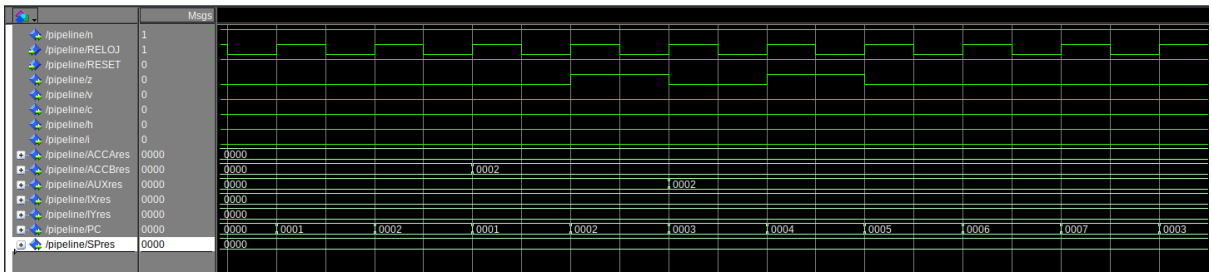

Figura 14 Simulación –Instrucciones donde se involucra el acumulador A y B

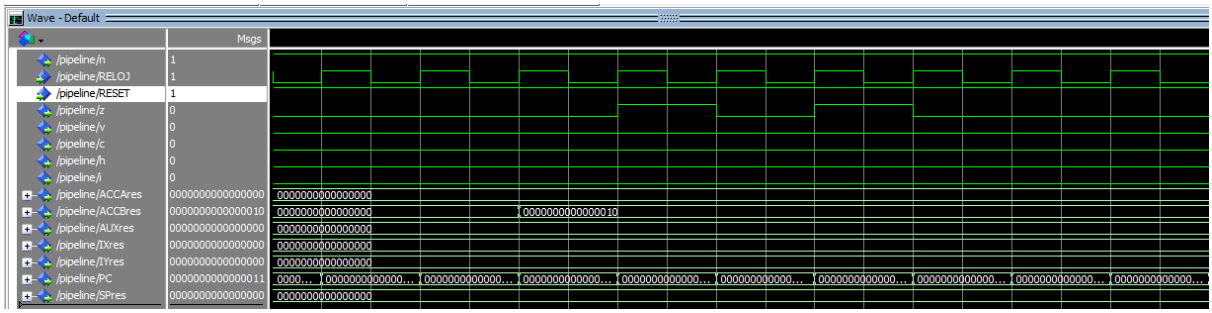

Figura 14.1 Simulación –Instrucciones donde se involucra el acumulador B

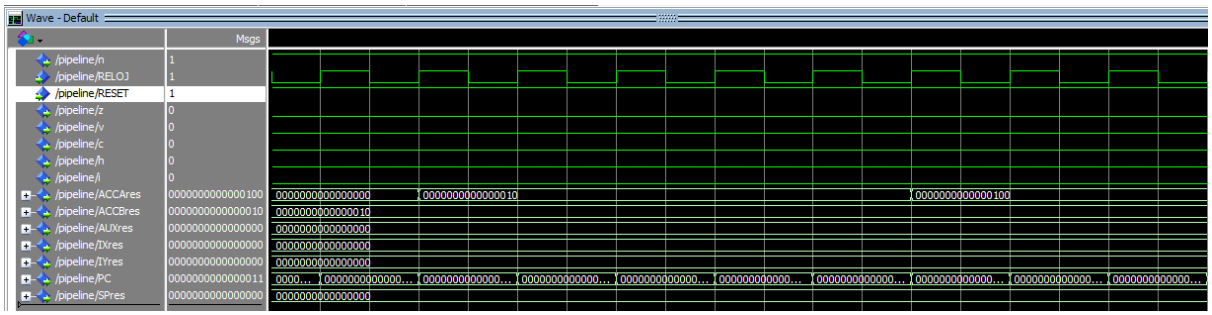

Figura 14.2 Simulación –Instrucciones donde se involucra el acumulador A

Como podemos observar en las simulaciones, se ve como el contador PC va incrementando en cada ciclo de reloj, asi mismo observamos como en el acumulador A y B se van cargando los valores correspondientes que se le asignaron. Y dado que no se hace ninguna operación o cambia alguna bandera con las instrucciones antes mencionadas, ninguna de estas dichas banderas altera su valor

Fuentes de Consulta

- Savage, J (2015) Diseño de microprocesadores. Facultad de Ingeniería. Universidad Nacional Autónoma de México. 482pp,
- Doblado, A (S.F) Microcontrolador mc68hc11 fundamentos, recursos y programación. Consultado el 06 de enero del 2021 de: http://www.iearobotics.com/proyectos/libro6811/libro-6811.pdf
- Anónimo (S.F) RISC. Consultado el 06 de enero del 2021 de: https://www.ecured.cu/RISC