Objetivo

Diseñar un microprocesador CISC de 8 bits, específicamente un 'clon' del microprocesador 68HC11 de Motorola.

Introducción

Microprocesador 68HC11

Motorola describe al 68HC11 como un microcontrolador de 8−bits fabricado con tecnología HCMOS, con una frecuencia de bus de 2 Mhz y con una amplia lista de recursos internos. Es capaz de ejecutar todas las instrucciones del M6800 y M6801 y 91 más que se le han incorporado.

Los recursos internos del microcontrolador son:

- 256 bytes de memoria RAM
- 5 puertos de 8 bits, con pines de entrada, salida y de entrada/salida
- Conversor analógico−digital de 8 canales y 8 bits de resolución.
- Una UART para comunicaciones serie asíncronas (SCI)
- Un módulo de comunicaciones serie síncronas (SPI)
- 5 comparadores con salida hardware
- 3 capturadores de entrada
- Un acumulador de pulsos externos de 8 bits
- Temporizador principal de 16 bits
- Interrupciones en tiempo real
- 2 entradas de interrupciones externas
- Software en ROM para cargar un programa externo en la RAM interna

Arquitectura del microprocesador 68HC11

Si se desea que el microprocesador ejecute un conjunto de instrucciones en lenguaje ensamblador, será necesario codificar cada instrucción en varias operaciones, de manera que sean totalmente entendibles para el microprocesador. La metodología a seguir son las máquinas de estados. Por lo tanto, para cada instrucción en ensamblador existirá un algoritmo de máquina de estados, que activará o desactivará secuencialmente, las líneas de control de la arquitectura.

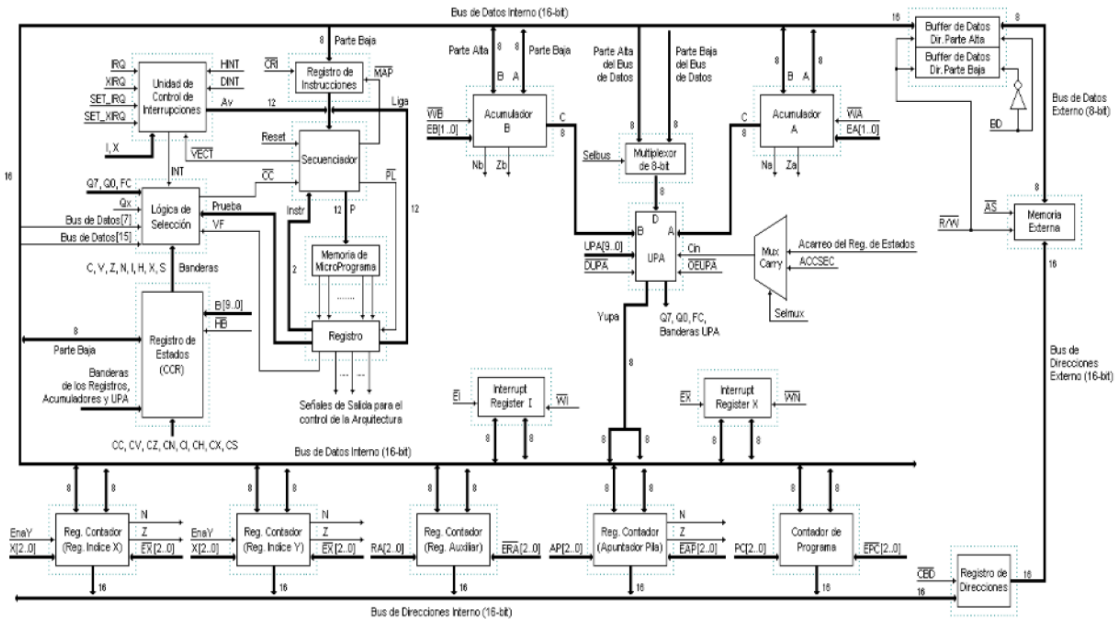

Figura 1 Arquitectura microprocesador 68HC11

Como se muestra en la figura anterior, se presenta el diagrama general de interconexión de la computadora. Usando como referencia esta figura, los pasos para ejecutar una instrucción en lenguaje ensamblador, residente en memoria externa, son los siguientes.

1) La UCP carga la dirección de la siguiente instrucción en el registro de direcciones, y se habilita la memoria para lectura. El contenido de la dirección seleccionada, con el código de la instrucción, es colocado en el bus de datos externo.
2) El código de la instrucción entra por el buffer de datos y se carga en el registro de instrucción.
3) La UCC decodifica la instrucción, es decir, salta a la dirección de microprograma dada por el código de la instrucción, en donde comienzan las micro-operaciones que serán ejecutadas.
4) Trae los operandos si así lo requiere la instrucción en ensamblador.
5) Si se trata de una operación lógico aritmética, se le indica a la UPA la operación a ejecutar.
6) Guarda el resultado en el lugar indicado por la instrucción en ensamblador y se actualizan las banderas o estados.
7) La UCP prepara la dirección de la siguiente instrucción a ejecutar, pero antes, la UCC revisa si hay interrupciones y efectúa el procedimiento de atención a interrupciones si es necesario.

La tarea de control será ejecutada por la UCC, quien activará las líneas de control de los distintos componentes de la arquitectura, de acuerdo a los algoritmos de máquinas de estados implantados. La activación de las líneas de control de la arquitectura se representa como salidas en un estado de una carta ASM.

Modos de direccionamiento

- *Acceso inmediato*

Las instrucciones que utilizan el acceso inmediato tienen el siguiente formato: el primer byte de la instrucción corresponde a su código de operación, y el segundo byte al valor de un dato de 8 bits.

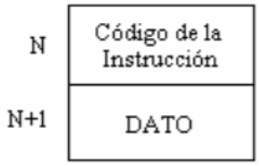

Figura 2.1 Acceso inmediato

- *Acceso Extendido*

Las instrucciones que utilizan el acceso extendido tienen el siguiente formato: el primer byte corresponde al código de operación de la instrucción, y los dos bytes siguientes a una dirección de 16 bits que contiene el valor del operando.

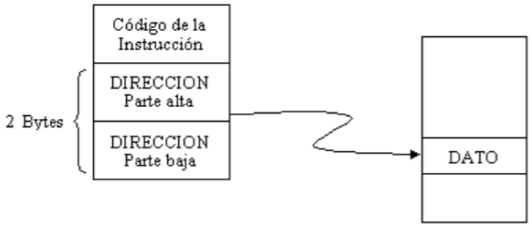

Figura 2.2 Acceso Extendido

- *Acceso Directo*

Las instrucciones que utilizan el acceso directo tienen el siguiente formato: el primer byte corresponde al código de operación de la instrucción, y el segundo byte a una dirección de 8 bits que contiene el valor del operando.

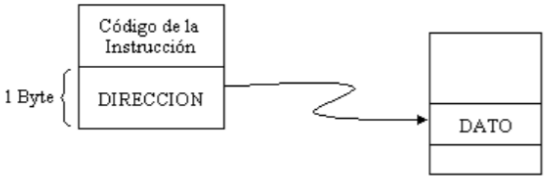

Figura 2.3 Acceso directo

- *Acceso Indexado*

Las instrucciones que utilizan el acceso indexado tienen el siguiente formato: el primer byte corresponde al código de operación de la instrucción, y el segundo byte a un desplazamiento de 8 bits sin signo, que se emplea para calcular la dirección del operando. La dirección del operando se calcula sumando el valor del desplazamiento más el contenido del registro X, ó el contenido del registro Y.

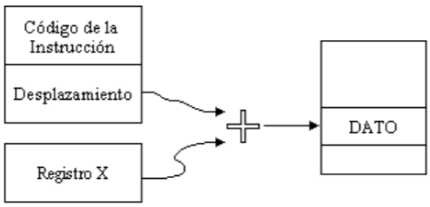

Figura 2.4 Acceso indexado

- *Acceso relativo*

Las instrucciones que utilizan el acceso relativo tienen el siguiente formato: el primer byte corresponde al código de operación de la instrucción, y el segundo byte a un desplazamiento de 8 bits con signo, que se emplea para calcular la dirección de la siguiente instrucción a ejecutar. Este tipo de acceso solo se utiliza en las instrucciones de salto. La dirección de salto se obtiene sumando el contenido del registro PC más el desplazamiento.

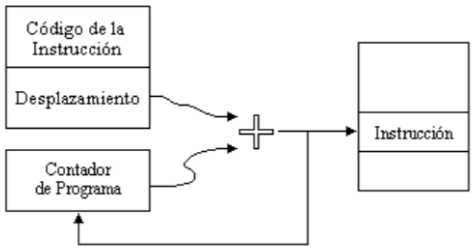

Figura 2.5 Acceso relativo

- *Acceso Inherente*

Este acceso no necesita operandos. El código de la instrucción es suficiente para saber el tipo de instrucción y la tarea que debe ejecutar

Desarrollo

Para poder construir en vhdl la descripción del procesador 68HC11, primeramente, construimos el bloque del Secuenciador básico, así como el de Registro de estados.

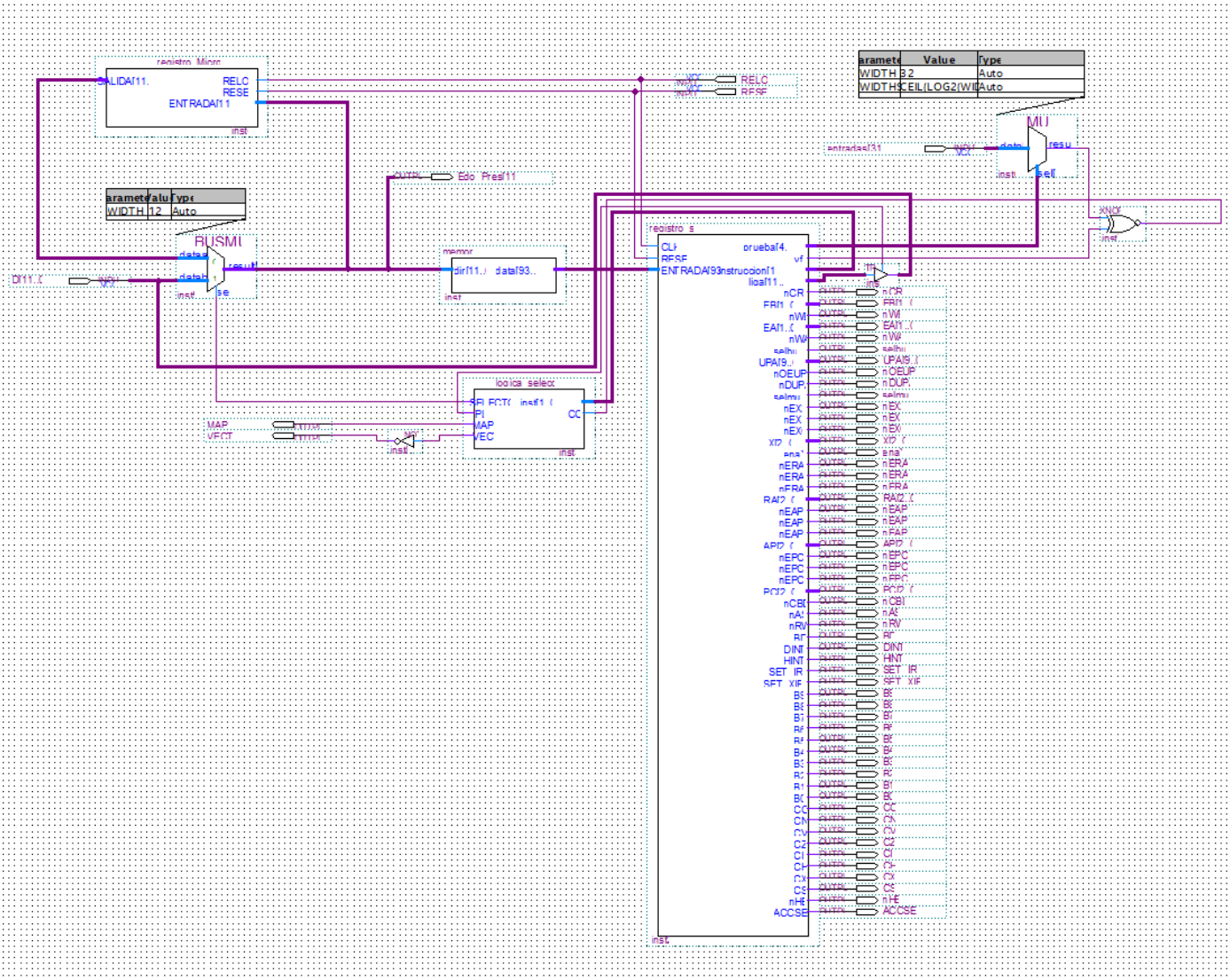

Figura 3 Diagrama interno del secuenciador básico

Como podemos observar en el diagrama del secuenciador básico sigue el esquema realizado en la practica anterior, teniendo el registro de PC, los multiplexadores, la lógica interna, etc. Agregando el registro de secuenciador el cual se conectará con los demás componentes del microprocesador.

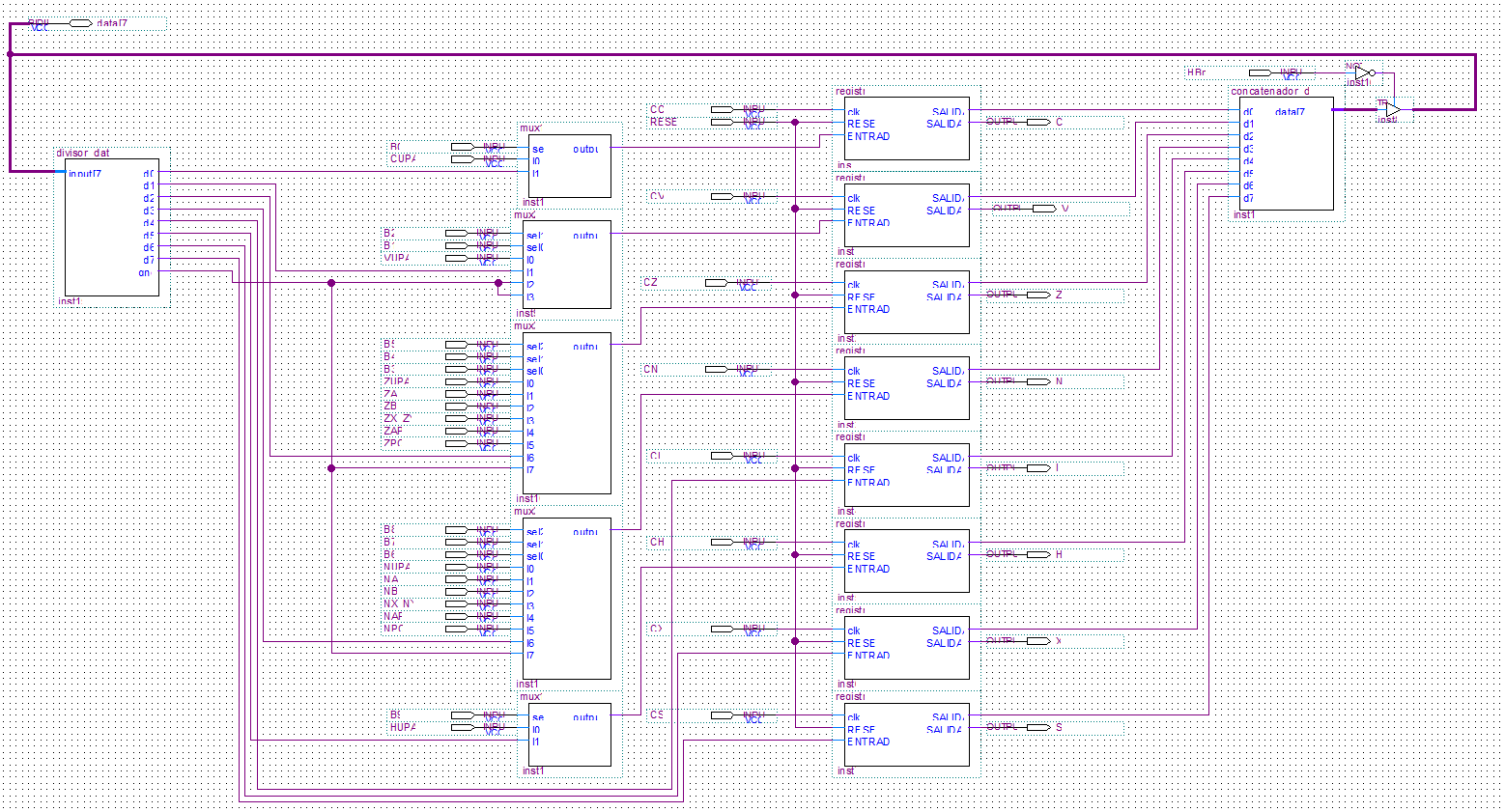

Figura 4 Diagrama interno del registro de estados

En el registro de estados (CCR) se encuentran las banderas de los registros, acumuladores y la UPA donde se ubican las líneas que controlan las operaciones.

Cada uno de estos componentes los compactamos haciéndolos uno solo respectivamente para poder asemejarnos más al esquema original del microprocesador 68HC11 que se mostró en la introducción. Teniendo así los siguientes bloques:

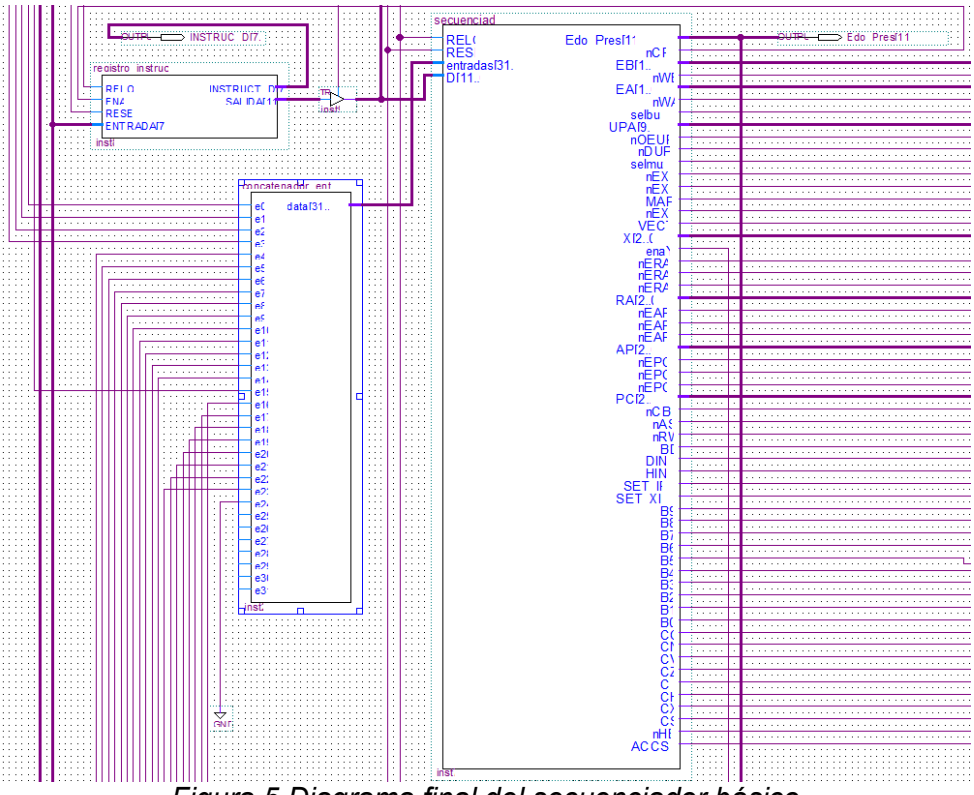

Figura 5 Diagrama final del secuenciador básico

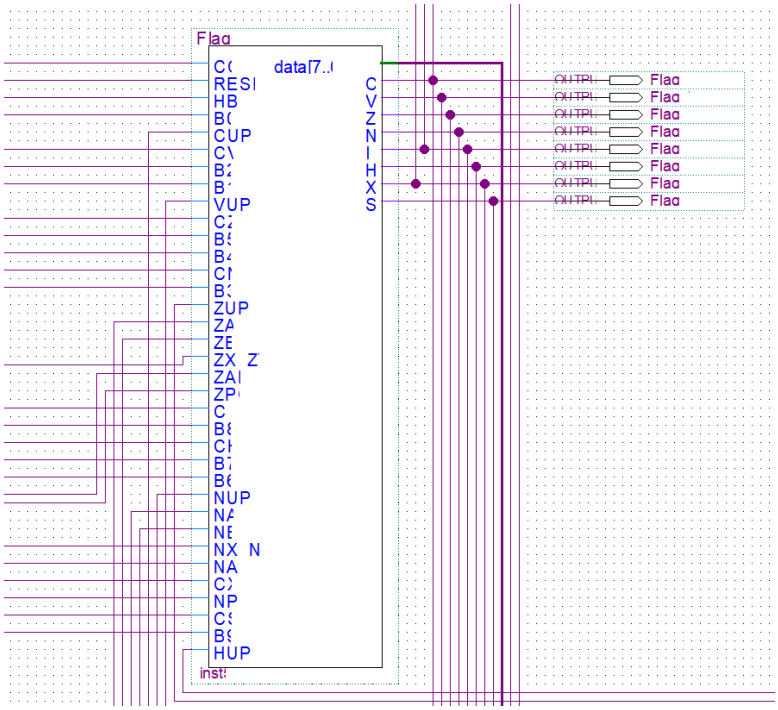

Figura 6 Diagrama final del registro de estados

Como podemos observar, ambos componentes se conectan a los demás componentes que se mostraron en la arquitectura del 68HC11, a continuación, se mostrara cada uno de los componentes que lo conforman.

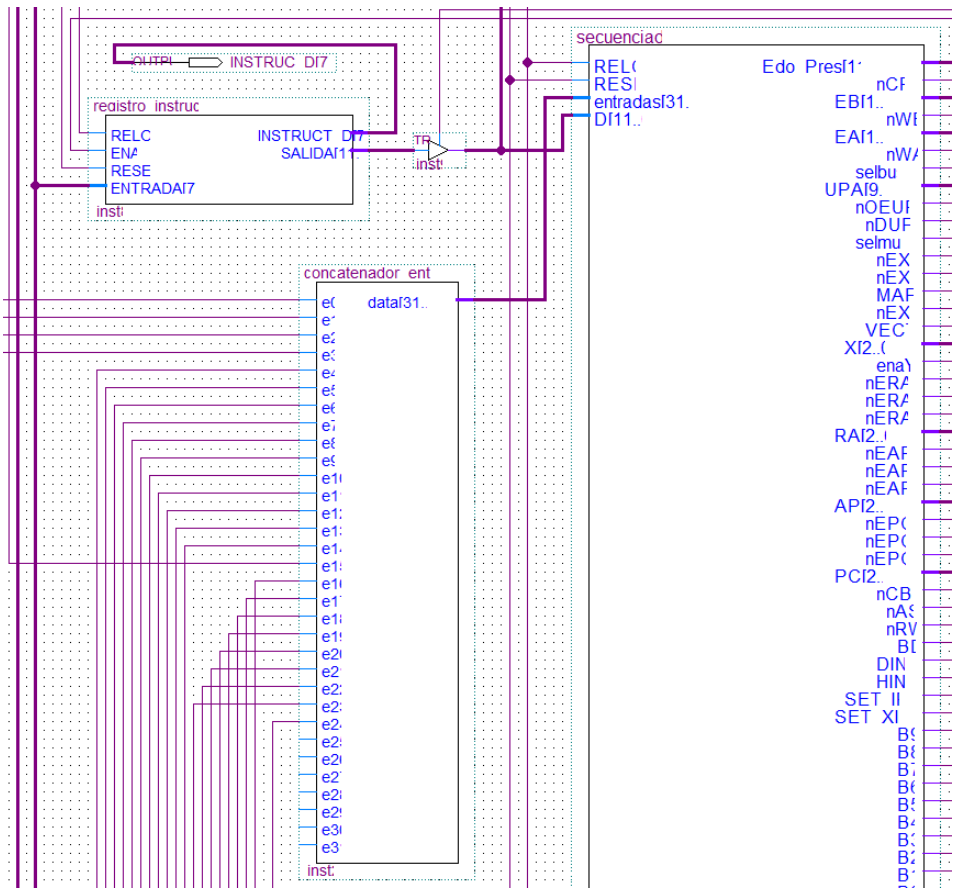

Figura 7 Conexión del secuenciador, registro de instrucciones y el concatenador de entradas

Primeramente, observamos que el secuenciador tiene como entrada el registro de instrucciones el cual se conecta directamente al Bus de Datos internos; y el concatenador de entradas al cual ingresan las salidas de los estados del registro de estados y a su vez se conecta a los contadores de los registros X, Y, Auxiliar, Pila y de PC.

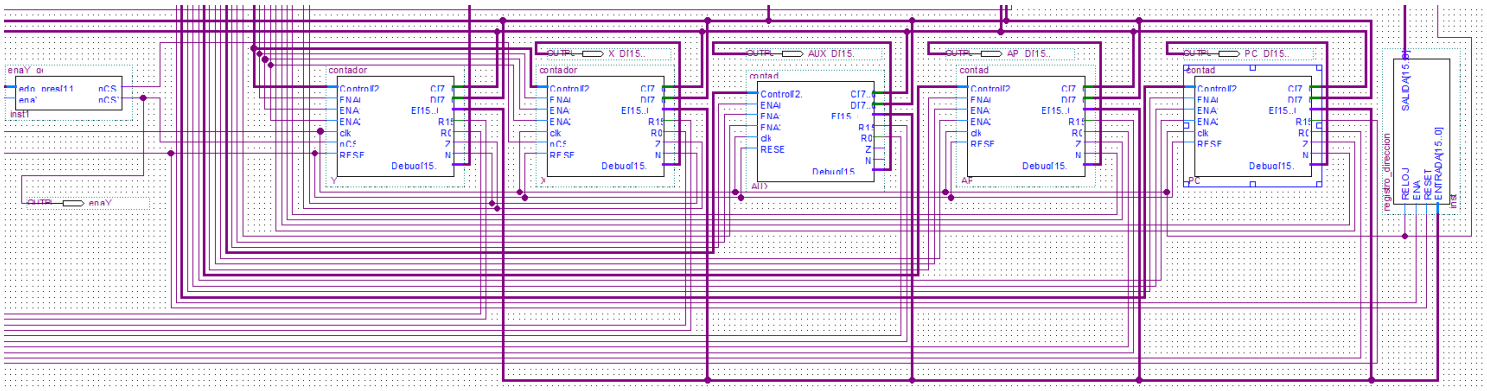

Figura 8 Conexiones de los Contadores

En la figura anterior se muestran los contadores que mencionamos anteriormente, estos se conectan al Bus de Datos Internos, al Bus de Direcciones Interno y cada uno de estos contadores se conectan a las correspondientes salidas del secuenciador que corresponden a cada uno de estos.

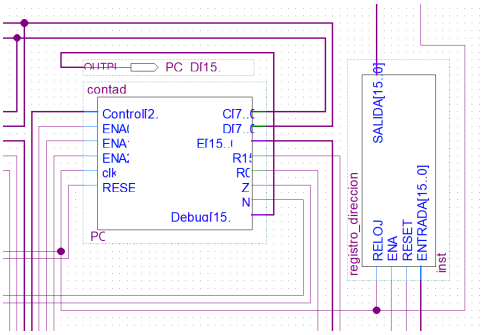

Figura 9 Conexión del Registro de direcciones

A lado de los contadores se encuentra el registro de direcciones que se conecta al Bus de Direcciones Interno, el cual dará como salida el Bus de Direcciones Externo el cual se conectará a la RAM.

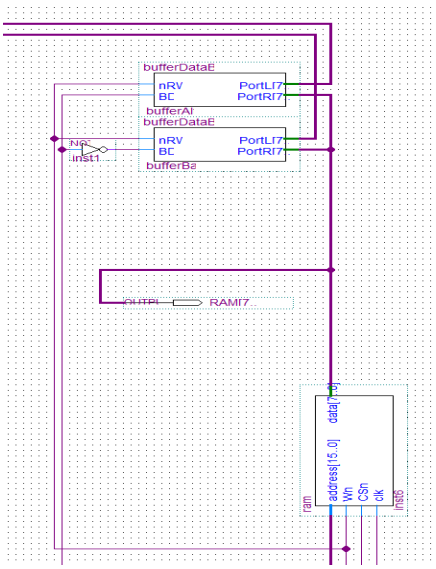

Figura 10 Conexión de la RAM con los buffers de datos (alta y baja)

Conectado a la salida de la RAM (Bus de Datos Externo) están los buffers de datos los cuales nos ayudaran a manejar de manera correcta el Bus de Datos Interno el cual es el principal Bus para el microprocesador 68HC11

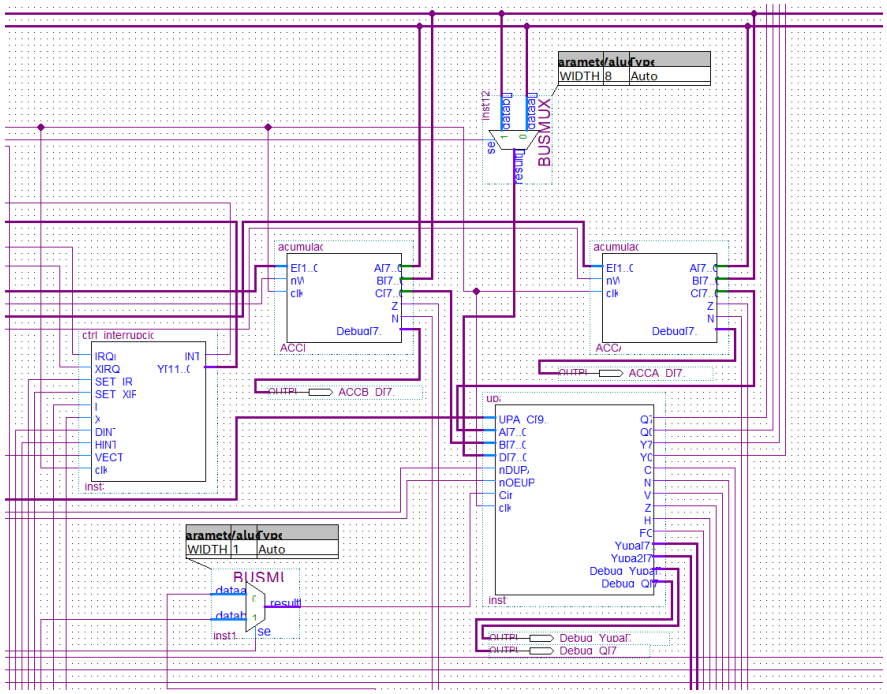

Figura 11 Conexión de la UPA con los registros A y B

Finalmente nos encontramos con los acumuladores A y B, los cuales se conectan al Bus de Datos Internos a la UPA y a las respectivas salidas del secuenciador; La UPA se conecta de igual forma al secuenciador y a su vez otorga las entradas al registro de estados; Por último, nos encontramos con el controlador de interrupciones el cual se conecta al secuenciador y al registro de estados.

Cartas ASM

Para poder llevar acabo las instrucciones que nosotros requerimos realice nuestro microprocesador, es necesario realizar dichas instrucciones en cartas ASM para obtener a detalle los bloques, banderas y microinstrucciones que ocupara, teniendo así las siguientes cartas ASM:

- *LDAB(DIR)*

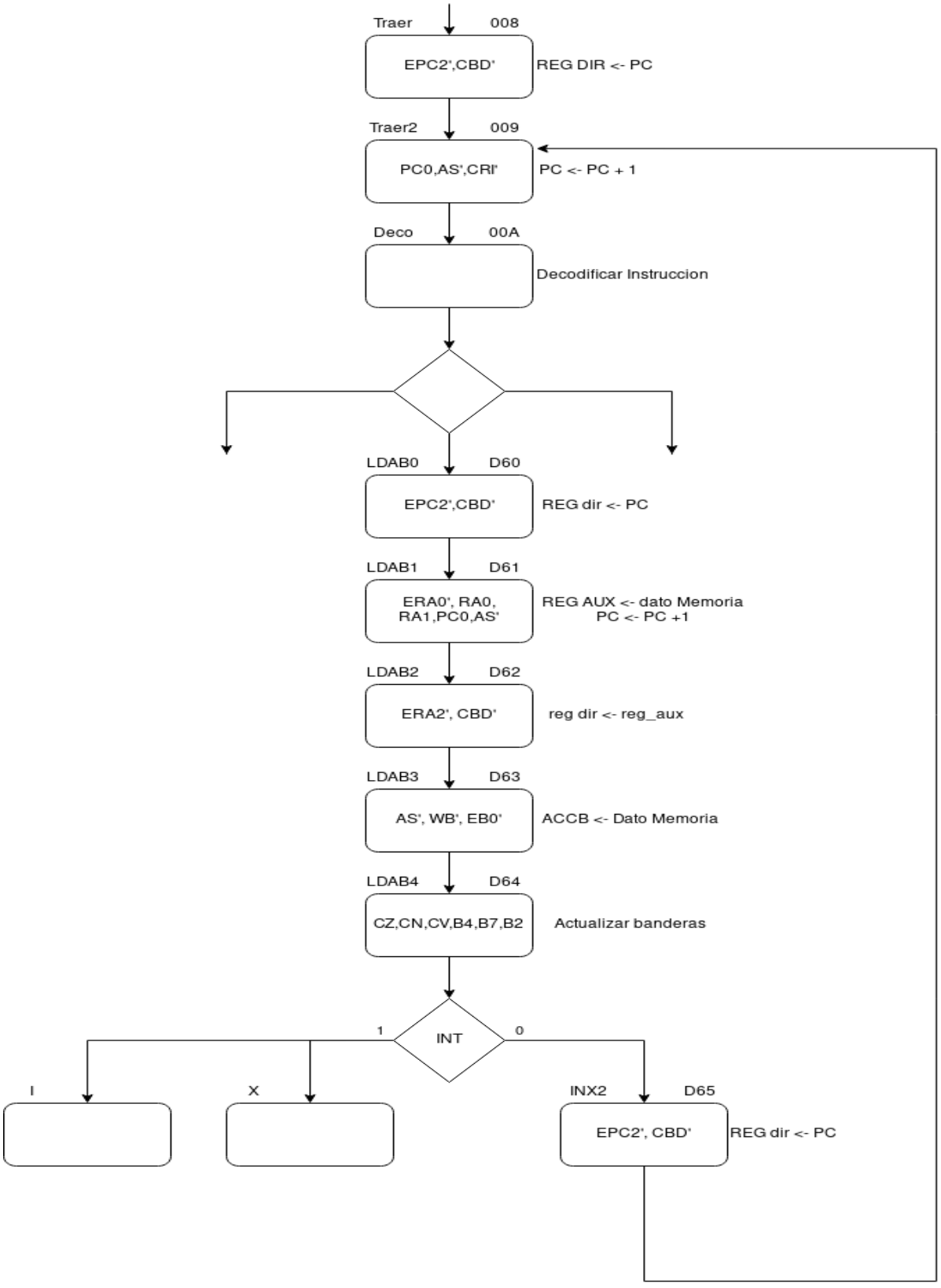

Figura 12.1 Carta ASM de la instrucción LDAB

- *LDAA(DIR)*

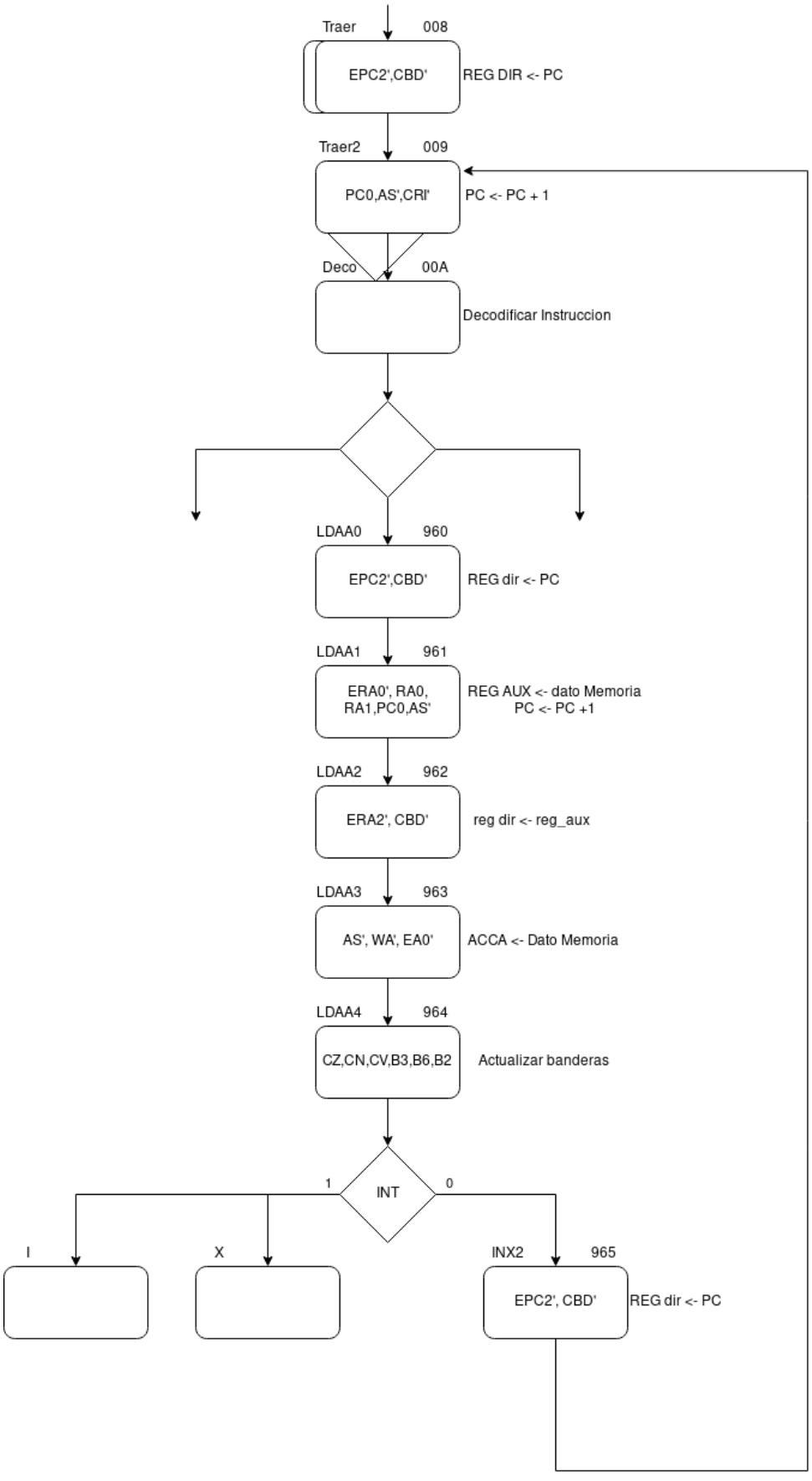

Figura 12.2 Carta ASM de la instrucción LDAA

- *ABA(INH)*

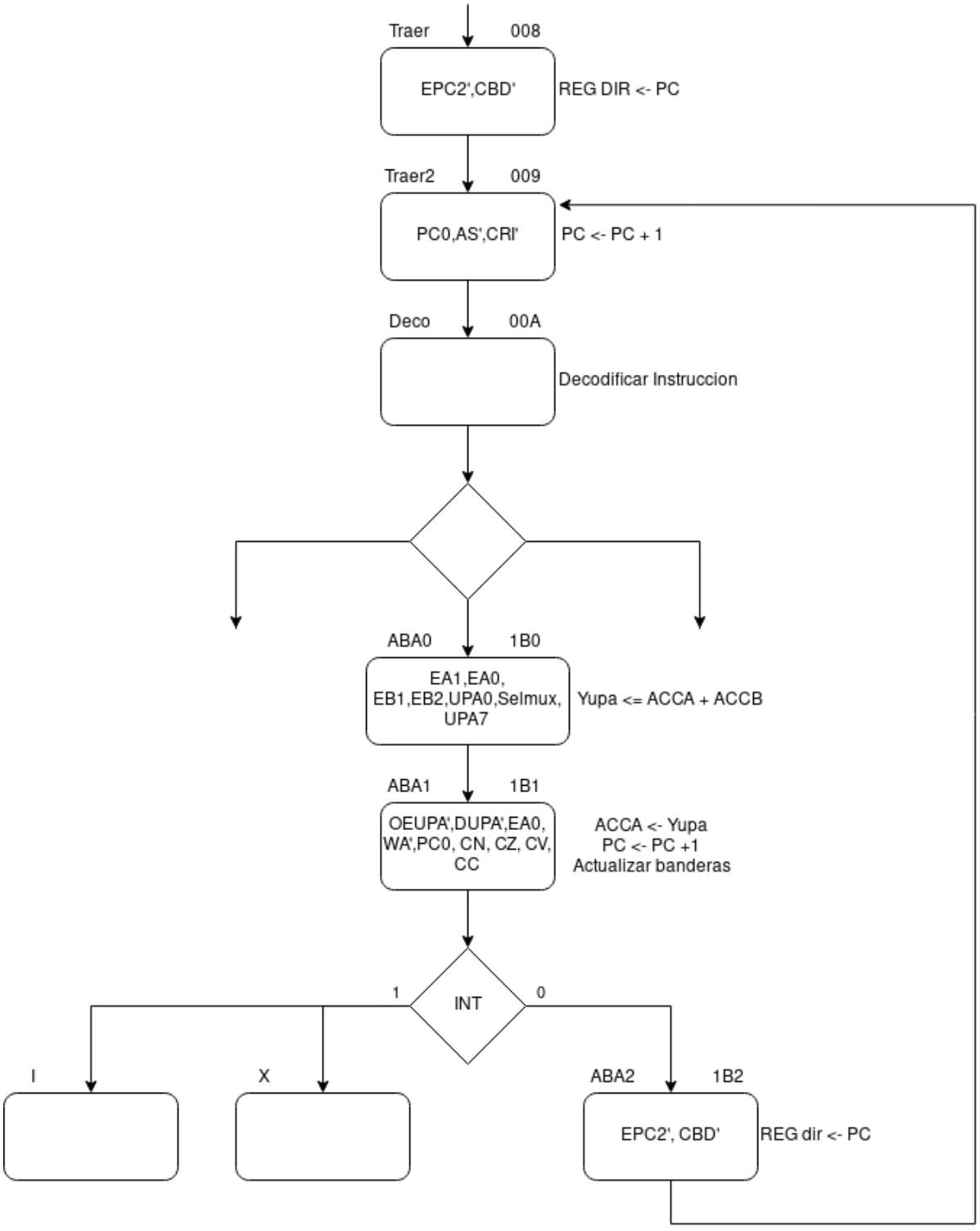

Figura 12.3 Carta ASM de la instrucción ABA

- *JMP(EXT)*

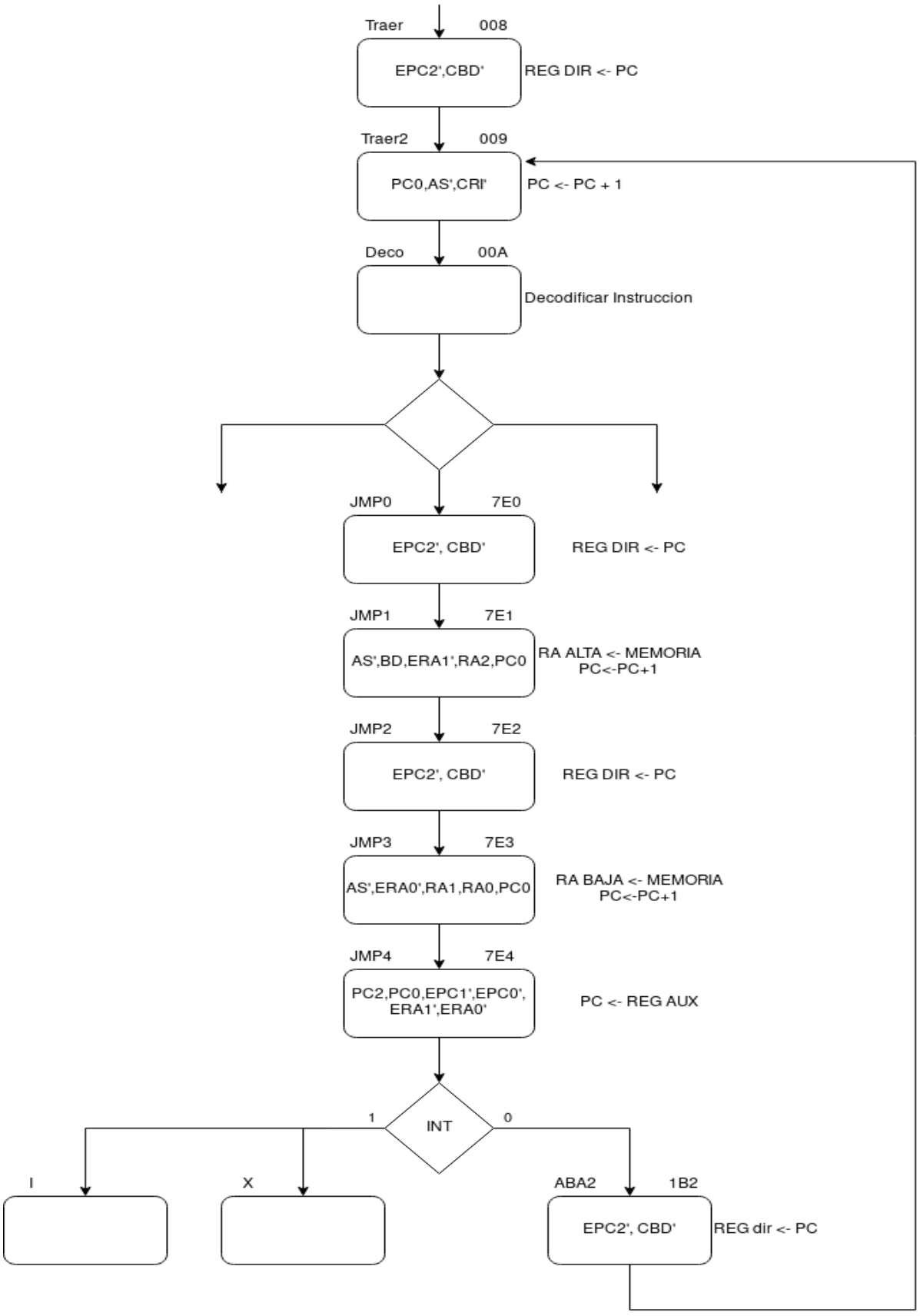

Figura 12.4 Carta ASM de la instrucción JMP

- *BRA*

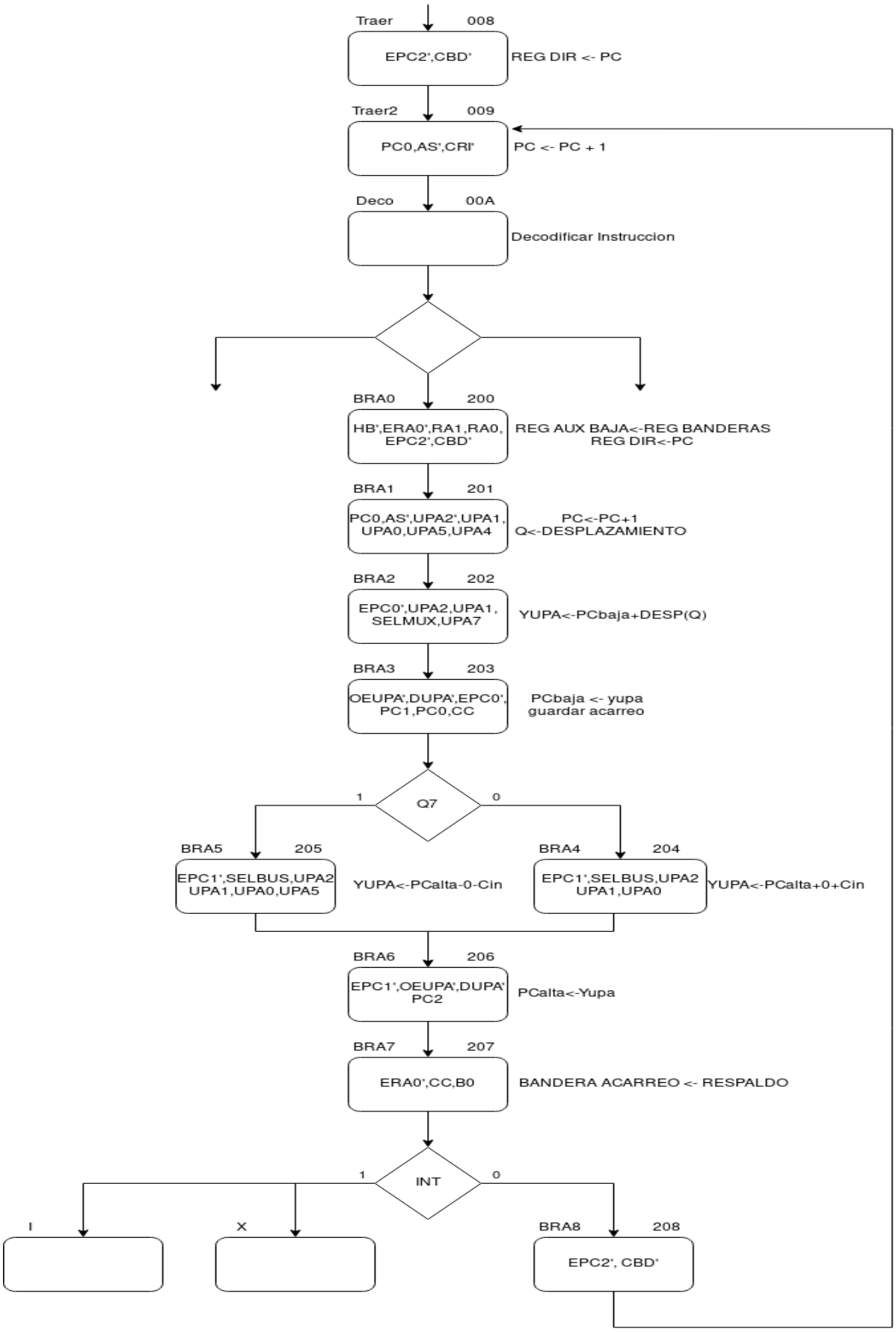

Figura 12.5 Carta ASM de la instrucción BRA

- *BNE*

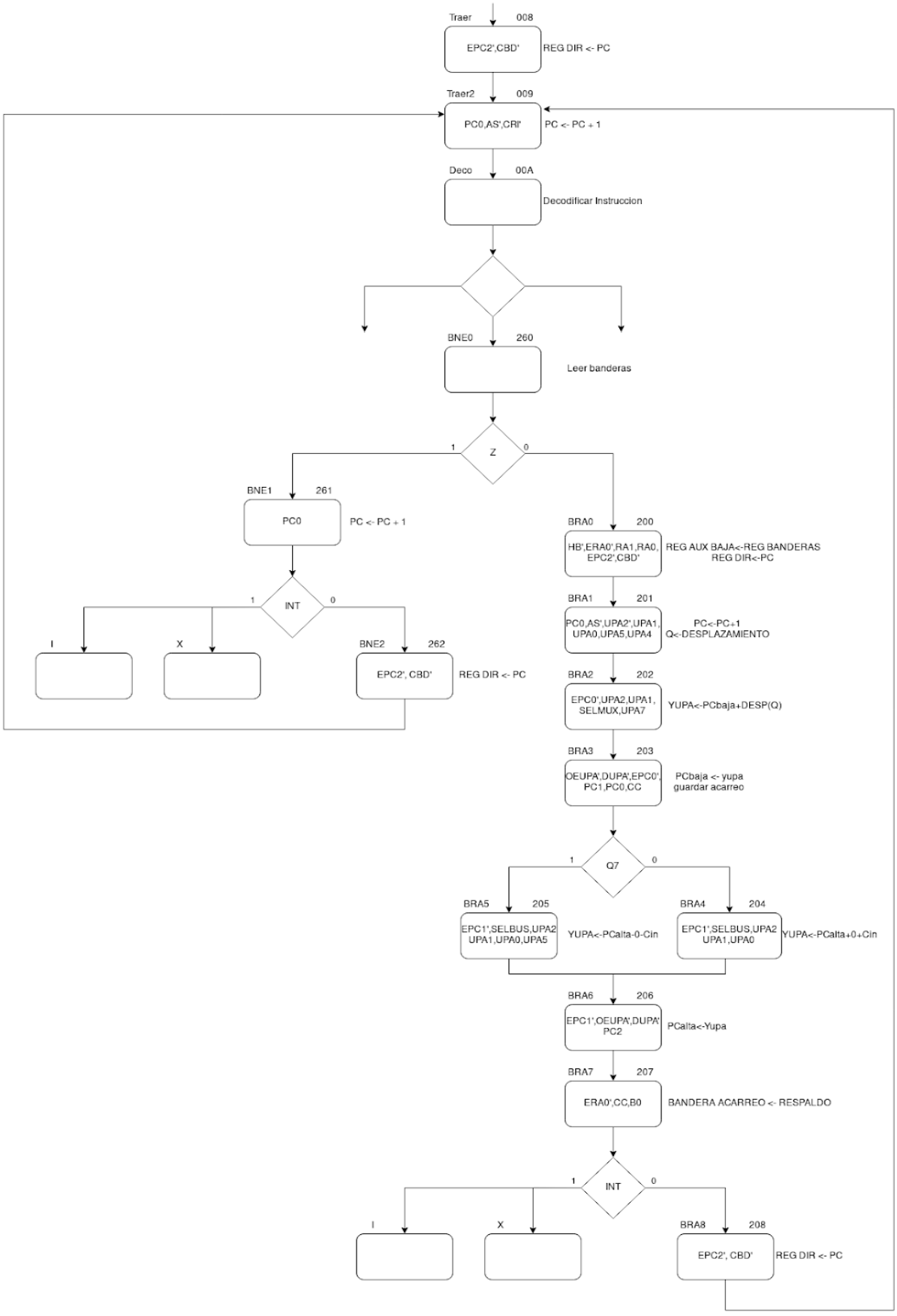

Figura 12.6 Carta ASM de la instrucción BNE

- *STAB*

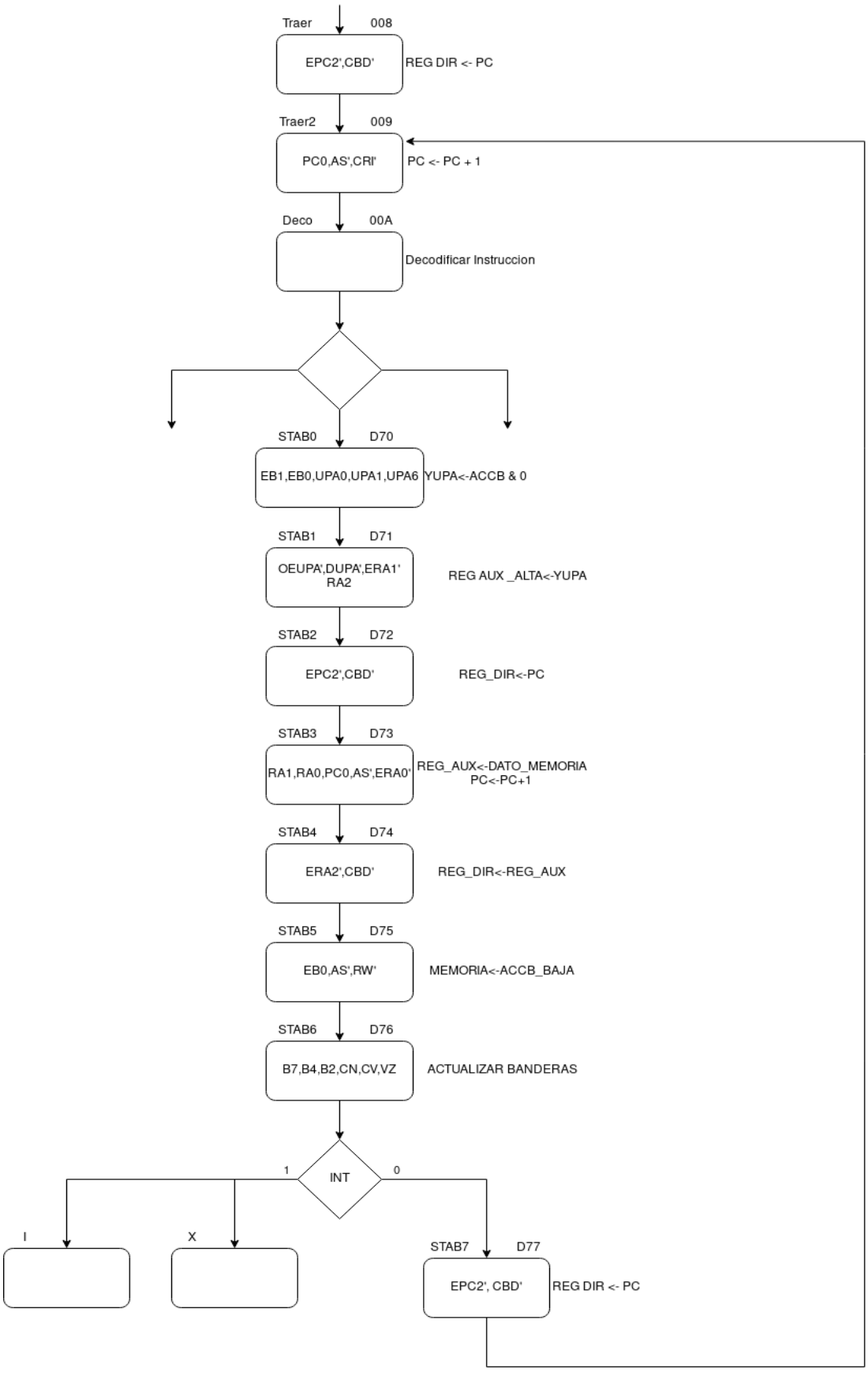

Figura 12.7 Carta ASM de la instrucción STAB

- *ADDA*

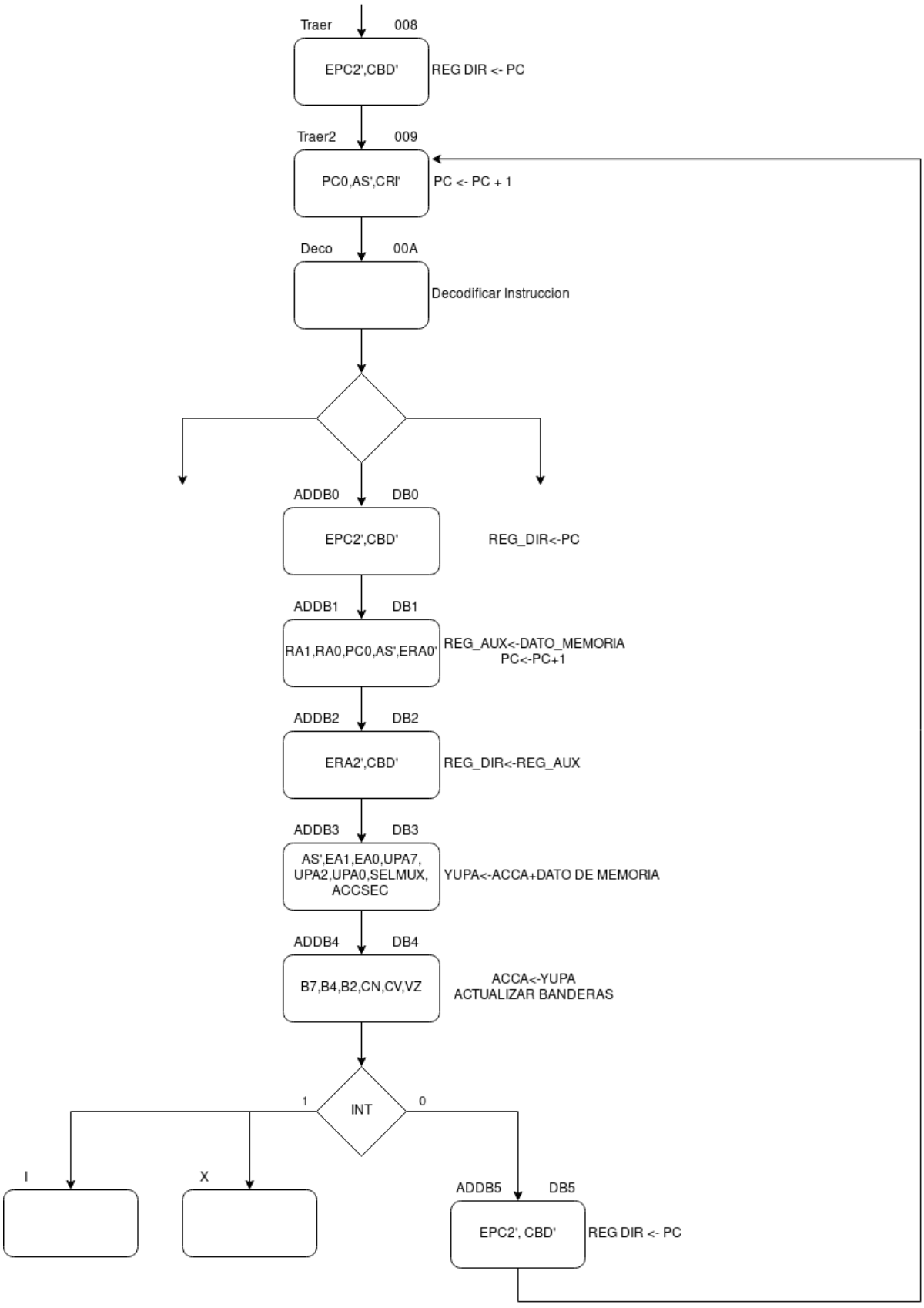

Figura 12.8 Carta ASM de la instrucción ADDA

Ejecución de las instrucciones

Una vez obtenida las cartas ASM de las instrucciones que queremos implementar, obtenemos la instrucción de forma binaria, siguiendo el siguiente orden:

P4 P3 P2 P1 P0 VF I1 I0 L11 L10 L9 L8 L7 L6 L5 L4 L3 L2 L1 L0 nCRI EB1 EB0 nWB EA1 EA0 nWA selbus UPA9 UPA8 UPA7 UPA6 UPA5 UPA4 UPA3 UPA2 UPA1 UPA0 nOEUPA nDUPA selmux nEX2 nEX1 nEX0 X2 X1 X0 EnaY nERA2 nERA1 nERA0 RA2 RA1 RA0 nEAP2 nEAP1 nEAP0 AP2 AP1 AP0 nEPC2 nEPC1 nEPC0 PC2 PC1 PC0 nCBD nAS nRW BD DINT HINT SET_IRQ SET_XIRQ B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 CC CN CV CZ CI CH CX CS nHB ACCSEC

Así, siguiendo las cartas ASM de las instrucciones encendimos las banderas o componentes que se necesitaran en cada bloque de proceso que se desgloso en las cartas ASM dentro de memory.vhd con la cual se empezaran a ejecutar las instrucciones dependiendo del orden en que se asignen.

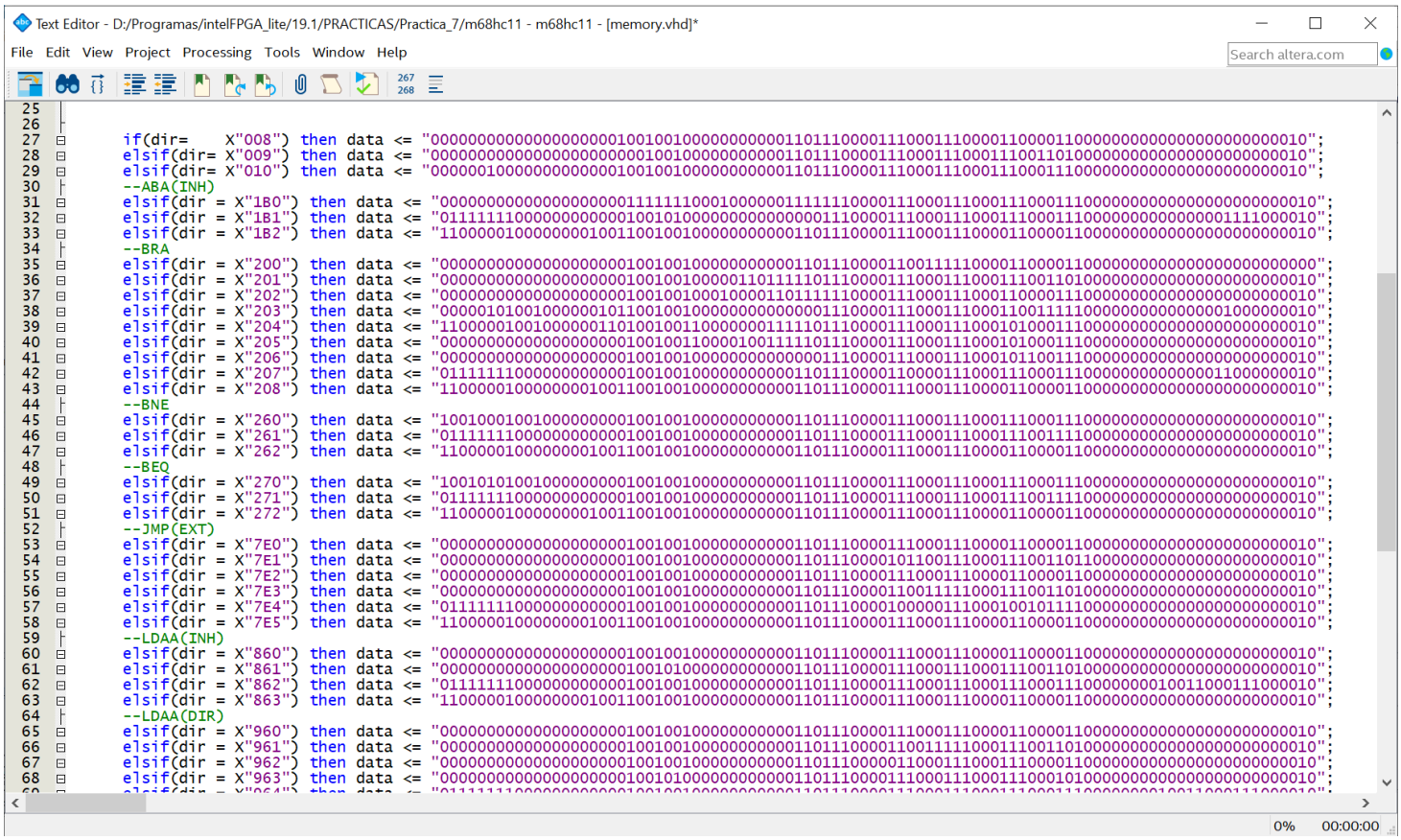

Figura 13.1 Instrucciones dentro de la memory.vhd

Con lo cual, a continuación, mostraremos el código de cada una de las instrucciones que se realizaron:

```
if(dir=    X"008") then data <=
"00000000000000000000010010010000000000011011100001110001110000110000110000000000000000000000000010";
       elsif(dir= X"009") then data <=
"00000000000000000000000100100000000000110111000011100011100001110011010000000000000000000000000010";
       elsif(dir= X"010") then data <=
"00000010000000000000010010010000000000011011100001110001110000111000111000000000000000000000000010";
       --ABA(INH)
       elsif(dir = X"1B0") then data <=
"00000000000000000000011111110001000001111111000011100011100001110001110000000000000000000000000010";
       elsif(dir = X"1B1") then data <=
"01111111000000000000100101000000000000000111000011100011100001110001110000000000000000001111000010";
       elsif(dir = X"1B2") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --BRA
       elsif(dir = X"200") then data <=
"00000000000000000000010010010000000000011011100001100111110000110000110000000000000000000000000000";
       elsif(dir = X"201") then data <=
"00000000000000000000010010010000011011111011100001110001110011010000000000000000000000000000000010";
       elsif(dir = X"202") then data <=
"00000000000000000000010010010001000110111111100001110001110001100001110000000000000000000000000010";
       elsif(dir = X"203") then data <=
"00000101001000000101110010010000000000000111000011100011100011001111100000000000000001000000010";
       elsif(dir = X"204") then data <=
"11000001001000000110100100110000000111110111000011100011100010100011100000000000000000000000000010";
       elsif(dir = X"205") then data <=
"00000000000000000000010010011000010011111011100001110001110001010001110000000000000000000000000010";
       elsif(dir = X"206") then data <=
"00000000000000000000010010010000000000000111000011100011100010110011100000000000000000000000000010";
       elsif(dir = X"207") then data <=
"01111111000000000000010010010000000000011011100001100001110001110000000000000000011000000010";
       elsif(dir = X"208") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --BNE
       elsif(dir = X"260") then data <=
"10010001001000000000100100100000000000110111000011100011100011100011100000000000000000000000000010";
       elsif(dir = X"261") then data <=
"01111111000000000000100100100000000000110111000011100011100011100111100000000000000000000000000010";
       elsif(dir = X"262") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --BEQ
       elsif(dir = X"270") then data <=
"10010101001000000000100100100000000000110111000011100011100011100011100000000000000000000000000010";
       elsif(dir = X"271") then data <=
"01111111000000000000100100100000000000110111000011100011100011100111100000000000000000000000000010";
       elsif(dir = X"272") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --JMP(EXT)
       elsif(dir = X"7E0") then data <=
"00000000000000000000010010010000000000011011100001110001110000110000110000000000000000000000000010";
       elsif(dir = X"7E1") then data <=
"00000000000000000000010010010000000000011011100001011001110001110011010000000000000000000000000010";
       elsif(dir = X"7E2") then data <=
"00000000000000000000010010010000000000011011100001110001110000110000110000000000000000000000000010";
       elsif(dir = X"7E3") then data <=
"00000000000000000000010010010000000000011011100001100111110001110011010000000000000000000000000010";
       elsif(dir = X"7E4") then data <=
"01111111000000000000100100100000000000110111000010000011100010010111100000000000000000000000000010";
       elsif(dir = X"7E5") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --LDAA(INH)
       elsif(dir= X"860") then data <=
"00000000000000000000010010010000000000011011100001110001110000110000110000000000000000000000000010";
       elsif(dir= X"861") then data <=
"00000000000000000000010010100000000000011011100001110001110001110011010000000000000000000000000010";
       elsif(dir= X"862") then data <=
"01111111000000000000100100100000000000110111000011100011100011100011100000000100110001110000010";
       elsif(dir= X"863") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --LDAA(DIR)
       elsif(dir = X"960") then data <=
"00000000000000000000010010010000000000011011100001110001110000110000110000000000000000000000000010";
       elsif(dir = X"961") then data <=
"00000000000000000000010010010000000000011011100001100111110001110011010000000000000000000000000010";
       elsif(dir = X"962") then data <=
"00000000000000000000010010010000000000011011100000110001110001110000110000000000000000000000000010";
       elsif(dir = X"963") then data <=
"00000000000000000000010010100000000000011011100001110001110001110001010000000000000000000000000010";
       elsif(dir = X"964") then data <=
"01111111000000000000100100100000000000110111000011100011100011100011100000000100110001110000010";
       elsif(dir = X"965") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000000010";
       --LDAB(INH)
```

```
       elsif(dir= X"C60") then data <=
"00000000000000000000100100100000000000110111000011100011100001100001100000000000000000000010";
       elsif(dir= X"C61") then data <=
"00000000000000000000101000100000000000110111000011100011100011101000000000000000000000000010";
       elsif(dir= X"C62") then data <=
"01111111000000000000100100100000000000110111000011100011100011100011100000001001010001110000010";
       elsif(dir= X"C63") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000010";
       --LDAB(DIR)
       elsif(dir = X"D60") then data <=
"00000000000000000000100100100000000000110111000011100011100001100001100000000000000000000000010";
       elsif(dir = X"D61") then data <=
"00000000000000000000100100100000000000110111000011001111100011100110100000000000000000000000010";
       elsif(dir = X"D62") then data <=
"00000000000000000000100100100000000000110111000011000111000111000011000000000000000000000000010";
       elsif(dir = X"D63") then data <=
"00000000000000000000101000100000000000110111000011100011100011100010100000000000000000000000010";
       elsif(dir = X"D64") then data <=
"01111111000000000000100100100000000000110111000011100011100011100011100000001001010001110000010";
       elsif(dir = X"D65") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000010";
       --STAB
       elsif(dir = X"d70") then data <=
"00000000000000000000111100100001000011110111000011100011100011100011100000000000000000000000010";
       elsif(dir = X"d71") then data <=
"00000000000000000000100100100000000000000111000010110011100011100011100000000000000000000000010";
       elsif(dir = X"d72") then data <=
"00000000000000000000100100100000000000110111000011100011100001100001100000000000000000000000010";
       elsif(dir = X"d73") then data <=
"00000000000000000000100100100000000000110111000011001111100011100110100000000000000000000000010";
       elsif(dir = X"d74") then data <=
"00000000000000000000100100100000000000110111000011000111000111000011000000000000000000000000010";
       elsif(dir = X"d75") then data <=
"00000000000000000000101100100000000000110111000011100011100011100010000000000000000000000000010";
       elsif(dir = X"d76") then data <=
"00000111000000000000100100100000000000110111000011100011100011100011100000001001010001110000010";
       elsif(dir = X"d77") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000010";
       elsif(dir = X"db0") then data <=
"00000000000000000000100100100000000000110111000011100011100001100001100000000000000000000000010";
       --ADDA
       elsif(dir = X"db1") then data <=
"00000000000000000000100100100000000000110111000011001111100011100110100000000000000000000000010";
       elsif(dir = X"db2") then data <=
"00000000000000000000100100100000000000110111000011000111000111000011000000000000000000000000010";
       elsif(dir = X"db3") then data <=
"00000000000000000000100111100010000101111111000011100011100011100010100000000000000000000000011";
       elsif(dir = X"db4") then data <=
"00000111000000000000100101000000000000000111000011100011100011100011100000000000000001111000010";
       elsif(dir = X"db5") then data <=
"11000001000000001001100100100000000000110111000011100011100001100001100000000000000000000000010";
```

Figura 13.2 Codigo completo de memory.vhd

Para la práctica se solicitaron las siguientes instrucciones:

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

Código para serie de fibonacci
LDAA #0x00
LDAB #0x01
STAA 0x04
STAB 0x05
ADDA 0x05
ADDB 0x04
BNE 0x02

Para lo cual se tienen las entradas, memoria del programa y la lógica del secuenciador siguientes:

- *Entradas*

Entrada	clave(BIN)	clave(hex)	clave(dec)
q7	00000	00	0
q0	00001	01	1
y7	00010	02	2
y0	00011	03	3
R15Y	00100	04	4
R0Y	00101	05	5
R15X	00110	06	6
R0X	00111	07	7
R15aux	01000	08	8
R0aux	01001	09	9
R15ap	01010	0A	10
R0ap	01011	0B	11
R15pc	01100	0C	12
R0pc	01101	0D	13
FC	01110	0E	14
INT	01111	0F	15
C	10000	10	16
V	10001	11	17
Z	10010	12	18
N	10011	13	19
I	10100	14	20
H	10101	15	21
X	10110	16	22
S	10111	17	23
AUX(GND)	11000	18	24

Figura 14 tabla de las entradas y su valor en binario, hexadecimal y decimal

Lógica del secuenciador

Entradas			Salidas			
I1	I0	CC	Selector	PL	MAP	VECT'
0	0	0	0	0	0	1
0	0	1	0	0	0	1
0	1	0	0	1	0	1
0	1	1	1	1	0	1
1	0	0	1	0	1	1
1	0	1	1	0	1	1
1	1	0	0	0	0	0
1	1	1	1	0	0	0

Figura 15 tabla de la lógica de las microinstrucciones del secuenciador

- *Contenido en memoria del programa para el programa de la práctica*

DIRECCIÓN (HEX)	CONTENIDO (HEX)
0x0000	C6
0x0001	02
0x0002	86
0x0003	00
0x0004	1B
0x0005	7E
0x0006	00
0x0007	04

Figura 16.1 Tabla del contenido de la memoria de para el programa de la práctica

- *Contenido en memoria del programa para serie de Fibonacci*

DIRECCIÓN (HEX)	CONTENIDO (HEX)
0x0000	86
0x0001	00
0x0002	C6
0x0003	01
0x0004	97
0x0005	04
0x0006	D7
0x0007	05
0x0008	9B
0x0009	05
0x0010	DB
0x0011	04
0x0012	26
0x0013	04

Figura 16.2 Tabla del contenido de la memoria de para serie de fibonacci

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*), la dirección que se sigue (*ADDRESS*), los registros AP (*AP*), el registro auxiliar (*AUX*), registro Q y Y (*Debug_Q y Debug_Y*), el estado presente(*Edo_Presente*), la instrucción que se está ejecutando (*INSTRUC*), el valor de la PC (*PC*), la RAM (*RAM*) y los valores del registro X y Y (*X_D y Y_D*)

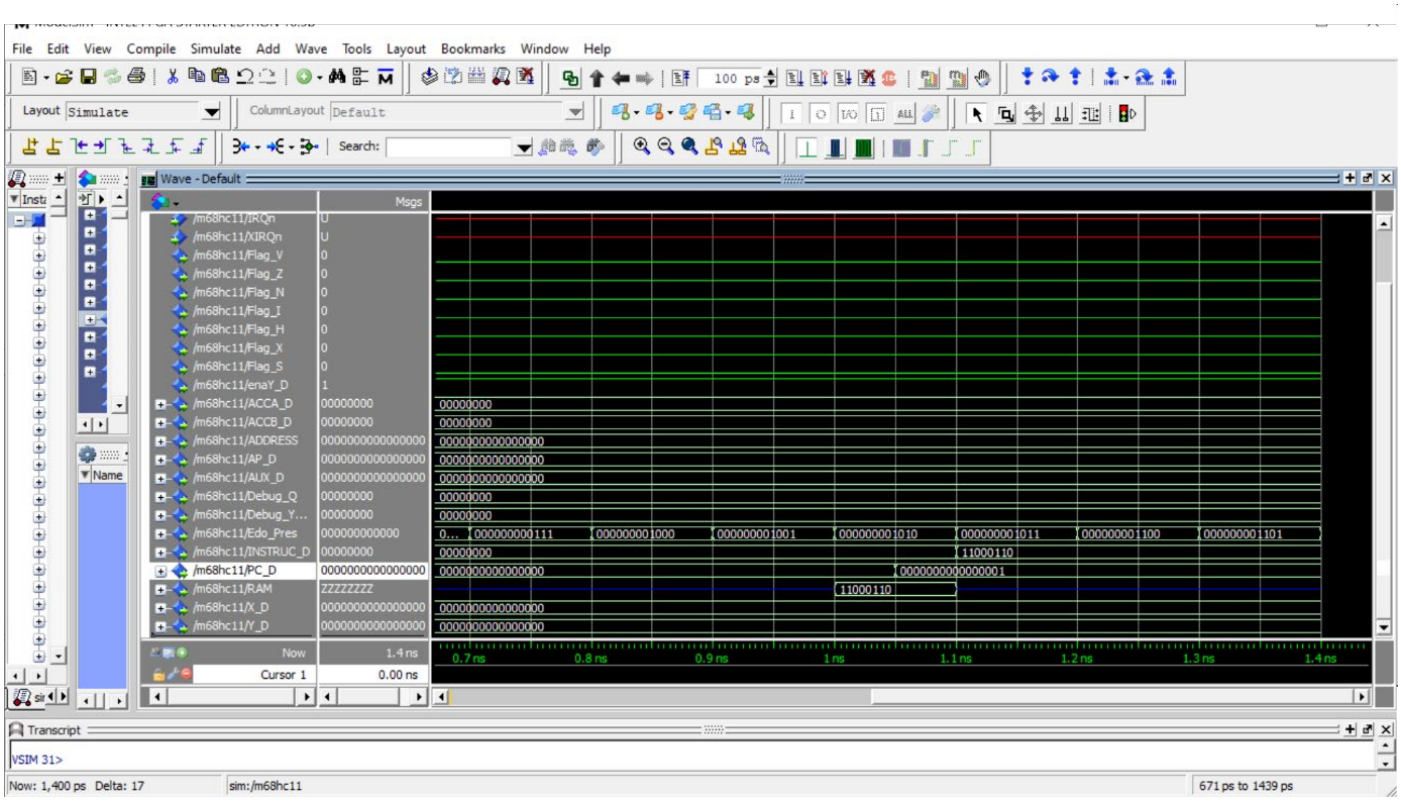

Figura 17.1 Simulación 01 – Primeras instrucciones, datos en el estado presente, instrucciones, PC y RAM

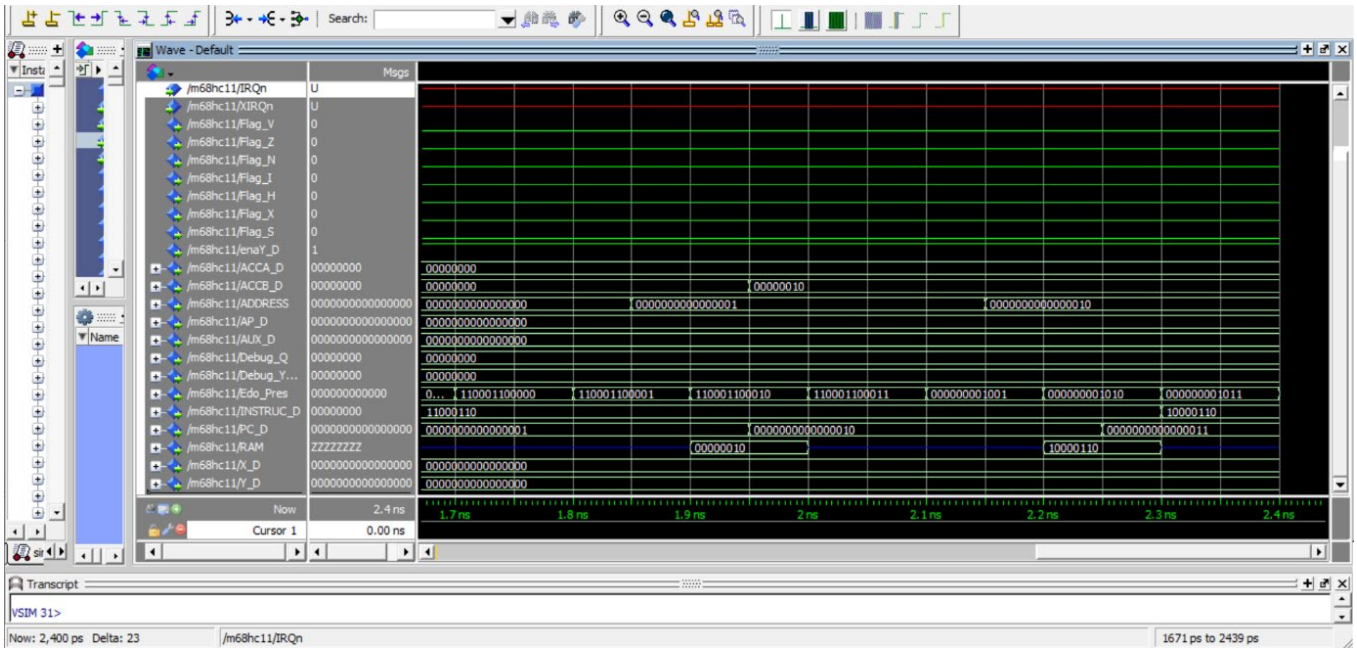

Figura 17.2 Simulación 02 –Instrucciones donde se involucra el acumulador B de 1.7ns a 2.4ns

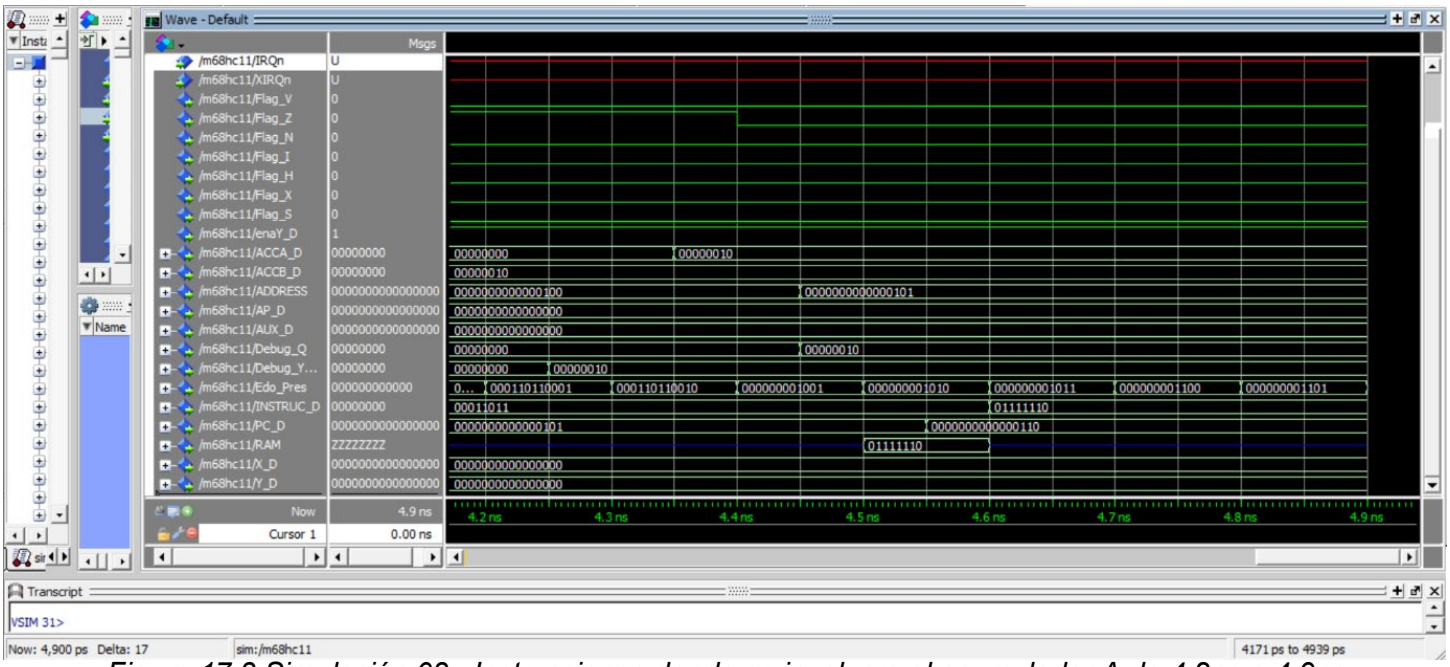

Figura 17.3 Simulación 03 –Instrucciones donde se involucra el acumulador A de 4.2ns a 4.9ns

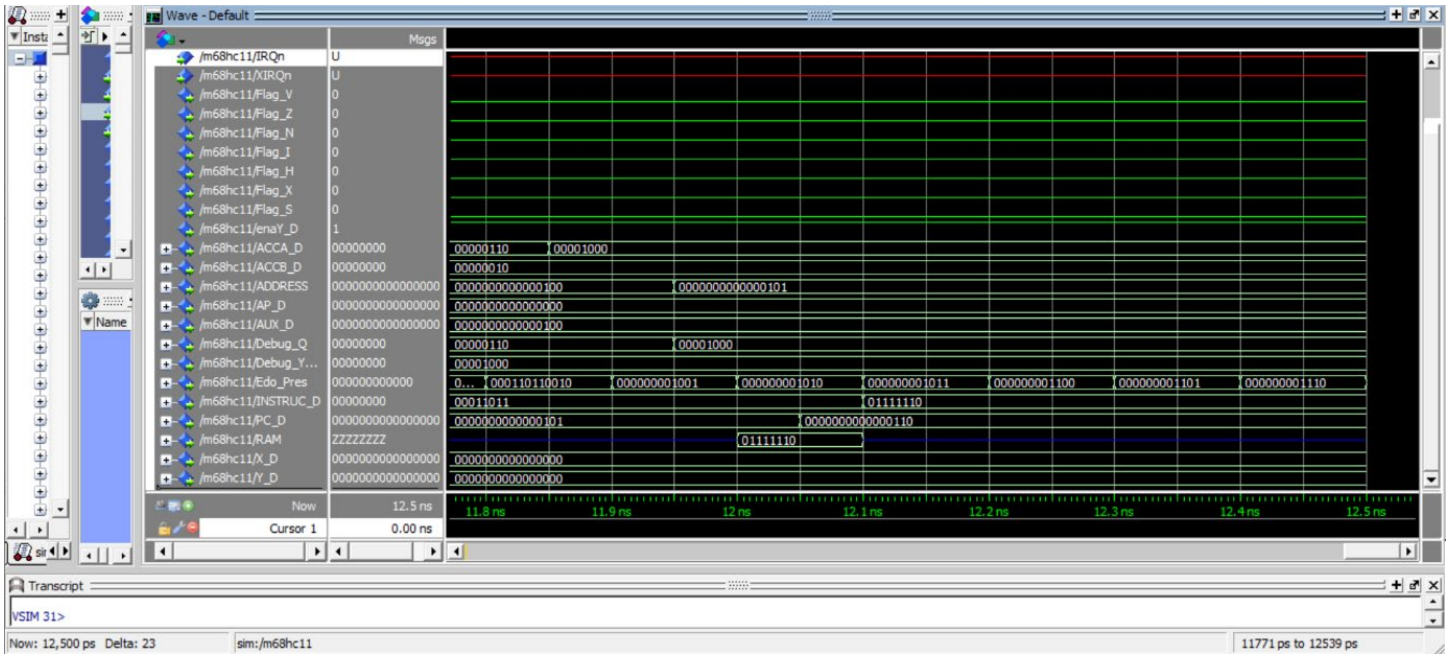

Figura 17.4 Simulación 04 de 11.8ns a 12.5ns

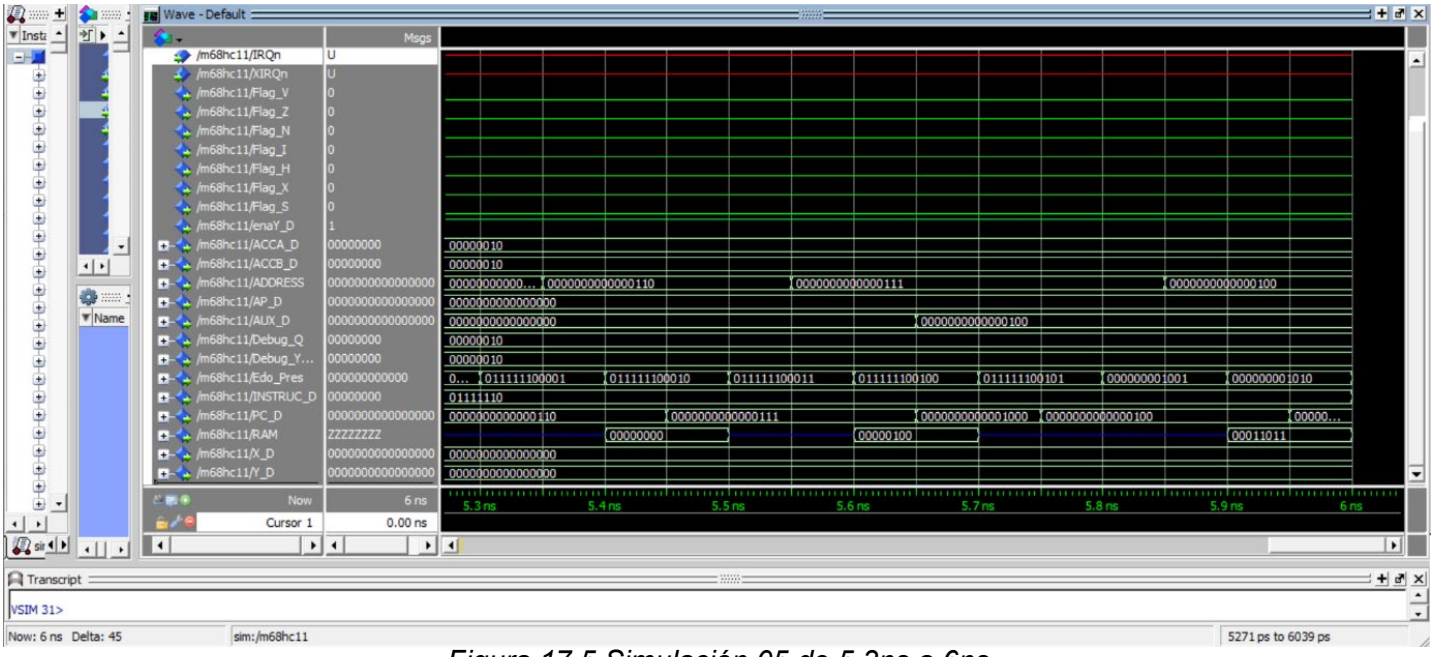

Figura 17.5 Simulación 05 de 5.3ns a 6ns

Como observamos con el paso del tiempo que está en ejecución se ejecutan y se almacenan diferentes valores tanto en los acumuladores como en los registros que se mencionaron anteriormente, siguiendo el orden que se estableció en el programa el cual estará ejecutándose continuamente dado que es la función principal que nosotros le asignamos, en este caso al microprocesador 68HC11.

Así mismo observamos que solo se hará uso de los registros o acumuladores en caso de que sea necesario, es por ello que el desglose de la instrucción en las cartas ASM es fundamental para poder hacer un uso correcto de todos los componentes que se armaron para poder creas la estructura CISC del 68HC11

Fuentes de Consulta

- Savage, J (2015) Diseño de microprocesadores. Facultad de Ingeniería. Universidad Nacional Autónoma de México. 482pp,
- Chávez, N (S.F) Construcción de máquinas de estados usando memorias. Consultado el 10 de diciembre del 2020 de: http://profesores.fi-b.unam.mx/normaelva/Direccionamientos.pdf
- Doblado, A (S.F) Microcontrolador mc68hc11 fundamentos, recursos y programación. Consultado el 10 de diciembre del 2020 de: http://www.iearobotics.com/proyectos/libro6811/libro-6811.pdf