CAPÍTULO VI
DISEÑO DE UN MICROPROCESADOR DE 8 BITS

6.1 ARQUITECTURA DEL MICROPROCESADOR 68HC11

En el capítulo anterior fueron diseñados los componentes básicos de una computadora. En este capítulo se muestra cómo hacer la interconexión de esos elementos, y la manera de controlarlos utilizando máquinas de estados.

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.

La figura 6.1 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. Recuerde que la activación de las líneas de control de la arquitectura se representan como salidas en un estado de una carta ASM.

A continuación se muestra la arquitectura del 68HC11 con los componentes desarrollados en el capítulo anterior. También se describe la función de las líneas de salida de la memoria de microprograma, líneas que controlan el funcionamiento de la arquitectura.

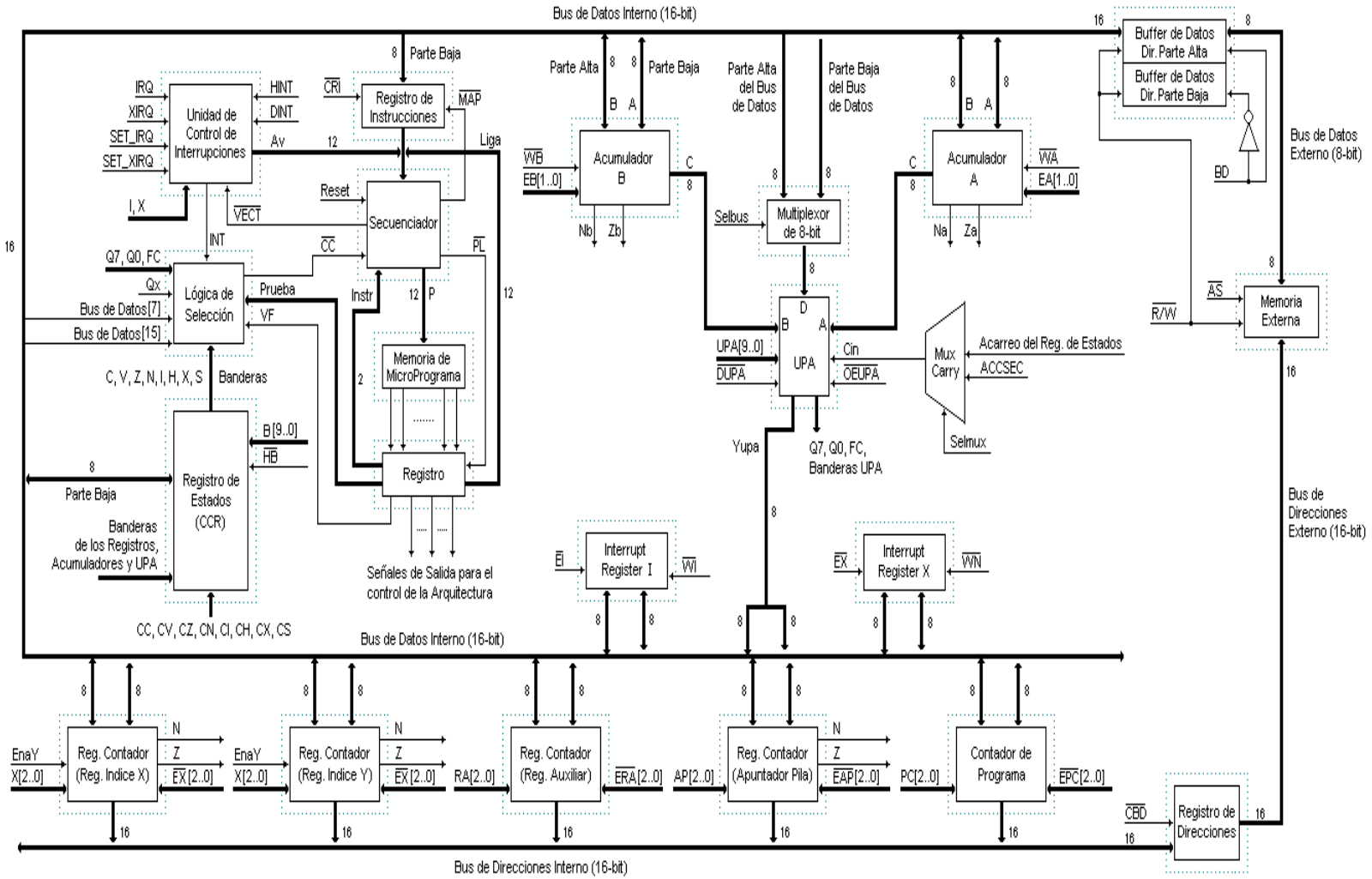


Figura 6.1. Arquitectura del microprocesador 68HC11 XE "Microprocesador 68HC11:Diagrama" .

Señales de Control	*Descripción*
CRI	Carga un dato en el registro de instrucciones.
EB1:EB0	Controlan las operaciones del acumulador B.
WB	Línea de escritura del acumulador B.
EA1:EA0	Controlan las operaciones del acumulador A.
WA	Línea de escritura del acumulador A.
Selbus	Selecciona la fuente de datos para la entrada D de la UPA. Si Selbus=0, el dato se toma de la parte baja del bus de datos interno, si no, de la parte alta.
UPA9:UPA0	Líneas de control de la UPA.
OEUPA	Habilita la salida de la UPA.
DUPA	Deshabilita el reloj interno de la UPA.
Selmux	Selecciona el carry del registro de estados o del secuenciador. El dato elegido representa el carry de entrada a la UPA.
EX2 : EX0	Seleccionan los buses del registro índice X ó del registro índice Y.
X2:X0	Controlan las operaciones del registro X ó del registro índice Y.
EnaY	Habilita las operaciones en los registros X e Y. Si EnaY=0, las operaciones en el registro X estarán habilitadas y en el registro Y no lo estarán. Si EnaY=1, las operaciones en el registro Y estarán habilitadas y en el registro X deshabilitadas.
ERA2 : ERA0	Seleccionan los buses del registro auxiliar RA.
RA2:RA0	Controlan las operaciones del registro auxiliar.
EAP2 : EAP0	Seleccionan los buses del registro apuntador de pila (AP).
AP2:AP0	Controlan las operaciones del registro AP.
EPC2 : EPC0	Seleccionan los buses del registro contador de programa (PC).
PC2:PC0	Controlan las operaciones del registro PC.
CBD	Carga un dato en el registro de direcciones.
WX	Carga un dato en el registro de interrupciones X.
EX	Habilita el registro de interrupciones X.
WI	Carga un dato en el registro de interrupciones I.
EI	Habilita el registro de interrupciones I.
AS	Habilita la memoria externa.
R/W	Señal de lectura/escritura de la memoria externa.
BD	Selecciona el buffer de datos que conecta al bus de datos externo con los buses de datos internos. Con un cero se selecciona la parte baja del buffer de datos, y con un uno la parte alta.
DINT, HINT, SET_IRQ, SET_XIRQ	Líneas que habilitan o deshabilitan la Unidad de Control de interrupciones.
B9:B0, CC, CN, CV, CZ, CI, CH, CX, CS	Líneas que controlan las operaciones en el registro de estados.
HB	Habilita el bus que conecta al registro de estados con el bus de datos interno.
I1:I0	Le indican al secuenciador qué tipo de instrucción ejecutar.
Prueba4:Prueba0	Seleccionan una variable de entrada en la lógica de selección.
VF	Línea de verdadero-falso.
ACCSEC	Acarreo proveniente de la memoria de microprograma. El valor de este

	acarreo puede ser modificado según nuestras necesidades.

Tabla 6.1. Descripción de las líneas de control para la arquitectura del 68HC11.

6.2 TIPOS DE INSTRUCCIONES DE ENSAMBLADOR DEL 68HC11

Las instrucciones en lenguaje ensamblador que puede ejecutar el microprocesador 68HC11, dependen de la forma en la que se acceden los datos. A continuación se explica brevemente, los seis tipos de acceso que existen.

6.2.1 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. Un ejemplo es la instrucción ADDA #Dato. Esta instrucción suma al acumulador A el contenido de la localidad de memoria siguiente al código de la instrucción. Note que el dato está precedido por el símbolo #, que se emplea para diferenciar este tipo de acceso de los demás.

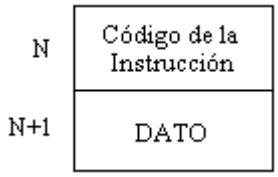


Figura 6.2. Acceso inmediato.

6.2.2 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. Un ejemplo es la instrucción ADDA Dirección_16_Bits. Esta instrucción suma al acumulador A el dato contenido en la localidad de memoria dada por *Dirección_16_Bits*.

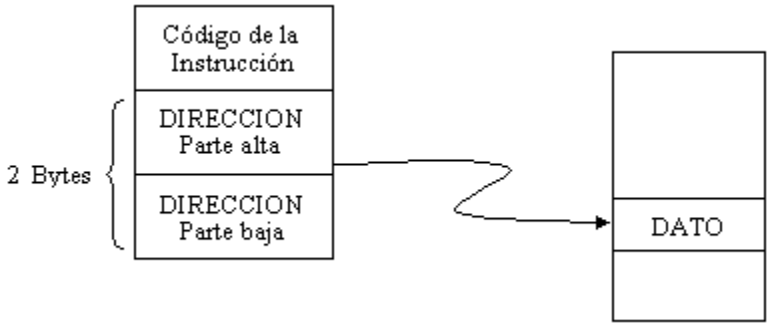


Figura 6.3. Acceso extendido.

6.2.3 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. Un ejemplo es la instrucción ADDA Dirección_8_Bits. Esta instrucción suma al acumulador A el dato contenido en la localidad de memoria dada por *Dirección_8_Bits*.

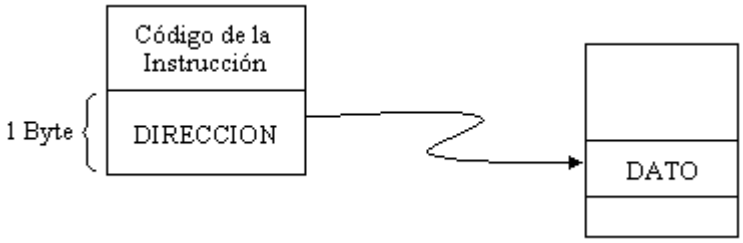


Figura 6.4. Acceso directo.

6.2.4 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. Un ejemplo es la instrucción ADDA Desplazamiento, X. Esta instrucción suma al acumulador A el dato contenido en la dirección (Registro X) + Desplazamiento.

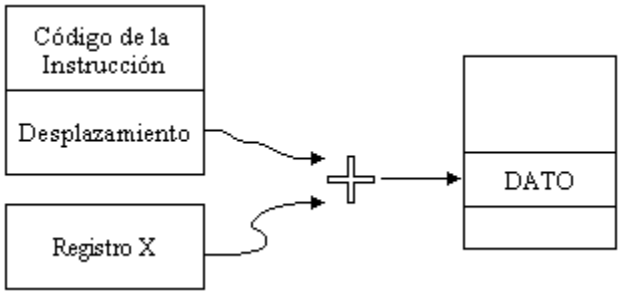


Figura 6.5. Acceso indexado.

6.2.5 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.

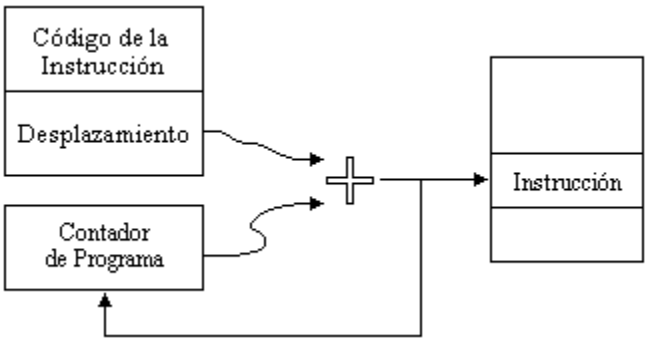


Figura 6.6. Acceso relativo.

6.2.6 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.

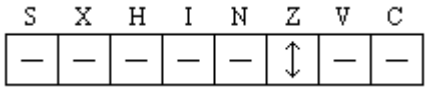
Ejemplo: INX. Incrementa el contenido del registro X en una unidad.

6.3 MICROPROGRAMACIÓN

A continuación se muestran algunos ejemplos de algoritmos de máquinas de estados en donde se utilizan los modos de acceso anteriores. Los algoritmos que se presentan hacen uso del modelo de arquitectura presentado en la figura 6.1, de manera que pueden ser ejecutados sobre ella.

6.3.1 INSTRUCCIÓN INX (Acceso Inherente)

Instrucción: INX
Operación: IX ⇐(IX) + 1
Código[2]: 08
Descripción: Incrementa el contenido del registro X en una unidad.
Banderas: El valor de la bandera de cero (Z) es actualizado tras la ejecución de esta instrucción. Z valdrá uno si el resultado en el registro índice X es cero, y valdrá cero en caso contrario. El estado de las demás banderas no se modifica.



La siguiente carta ASM representa el algoritmo que ejecuta la instrucción INX.

[2] El código de operación en el 68HC11 es una palabra única de 8 bits para cada instrucción en ensamblador, la cual la identifica del resto de las instrucciones. Estos códigos de operación se presentan en formato hexadecimal.

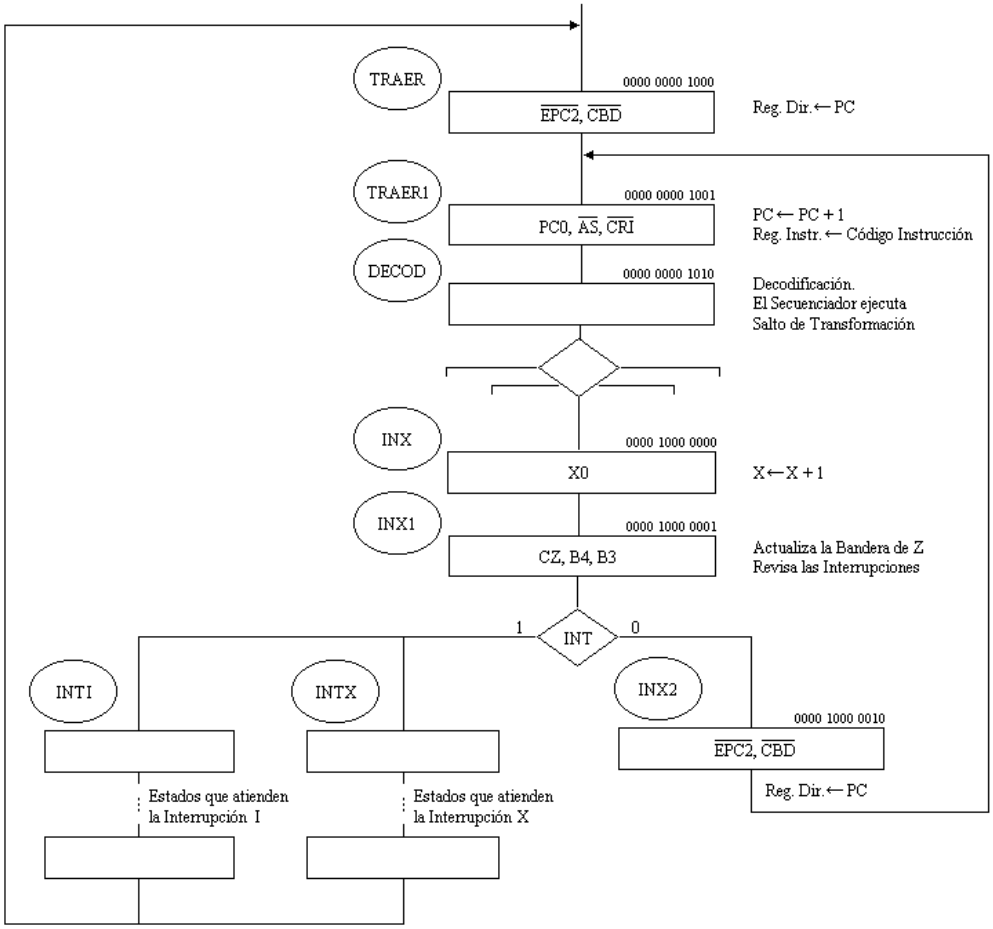


Figura 6.7. Carta ASM para la instrucción INX (acceso inherente).

A continuación se explica la carta ASM de la figura 6.7.

En los primeros estados se ejecuta el ciclo “FETCH”, es decir, se trae el código de operación de la siguiente instrucción a ejecutar. El contador de programa, PC, contiene la dirección en memoria en donde se localiza la siguiente instrucción a ejecutar, por lo tanto, se coloca el contenido de PC en el registro de direcciones; éste último está conectado al bus de direcciones de la memoria externa.

En el estado “TRAER” de la carta ASM se efectúan las actividades anteriormente descritas. Primero, se lee el contenido del registro PC, es por ello que las líneas PC2, PC1 y PC0 son puestas a ceros, es decir, se habilita al registro PC para sólo lectura. A continuación se habilita la salida del registro PC para que el dato contenido en él pase hacia el bus de direcciones interno, esto se realiza colocando la línea EPC2 [3] a cero. Finalmente, para cargar la dirección de PC en el registro de

[3] Véase la tabla de operación del Registro de 16 bits presentada en el Capítulo 5.

direcciones se coloca la señal CBD a cero[4]. La figura 6.8 muestra de manera gráfica lo mencionado con anterioridad.

En el estado “TRAER1” se lee de la memoria externa, el código de operación de la instrucción. La línea R/W puesta a uno habilita la lectura de datos en la memoria externa. En este caso la línea R/W no se escribe dentro del estado, se asume que vale uno puesto que está negada. También se activa la señal AS para habilitar a la memoria antes de efectuar la operación de lectura. El código de la instrucción, leído de memoria, viaja a través del bus de datos externo y continúa su camino hacia la parte baja del bus de datos interno. La señal BD puesta a cero y la señal R/W puesta a uno son suficientes para que el buffer de datos decida entre pasar un dato hacia la parte alta o hacia la parte baja del bus de datos interno. Una vez que el código de la instrucción está presente en el bus de datos, el registro de instrucción podrá guardarlo, para ello se coloca a cero a la señal CRI . De esta manera, en el registro de instrucción queda guardada una copia del código de la instrucción a ejecutar. Por último, la señal PC0=1 incrementa en una unidad el contenido del registro PC para así obtener la dirección de la siguiente palabra en memoria. Todas estas acciones son mostradas en la figura 6.9.

En el estado “DECOD” se decodifica la instrucción en ensamblador, es decir, se le indica al secuenciador a qué dirección saltar para iniciar las microinstrucciones que ejecutan dicha instrucción. En este estado, el secuenciador efectúa un salto de transformación activando la línea de MAP , la cual selecciona el contenido del registro de instrucción como el dato de entrada hacia la entrada D del secuenciador. La figura 6.10 muestra de manera gráfica lo mencionado con anterioridad.

En el registro de instrucción se tiene el código de la instrucción INX, es decir, 08 en formato hexadecimal. Dado que la entrada D del secuenciador es de 12 bits, los 8 bits del registro de instrucción deben extenderse a 12 bits, de manera que se colocan cuatro ceros en la parte menos significativa del contenido del registro de instrucción y así se forma la dirección de inicio de la instrucción INX: 0000 1000 0000.

[4] Recuerde que en notación de cartas ASM sólo se colocan las salidas activas. Por ejemplo, si la señal de salida está negada, colocar el nombre de la salida en un estado significará que la señal toma el valor de cero, en caso contrario, se asume que la señal vale uno. Por otra parte, si la señal de salida no está negada, colocar el nombre de la salida en un estado significará que la señal toma el valor de uno, si no, se asume que la señal vale cero.

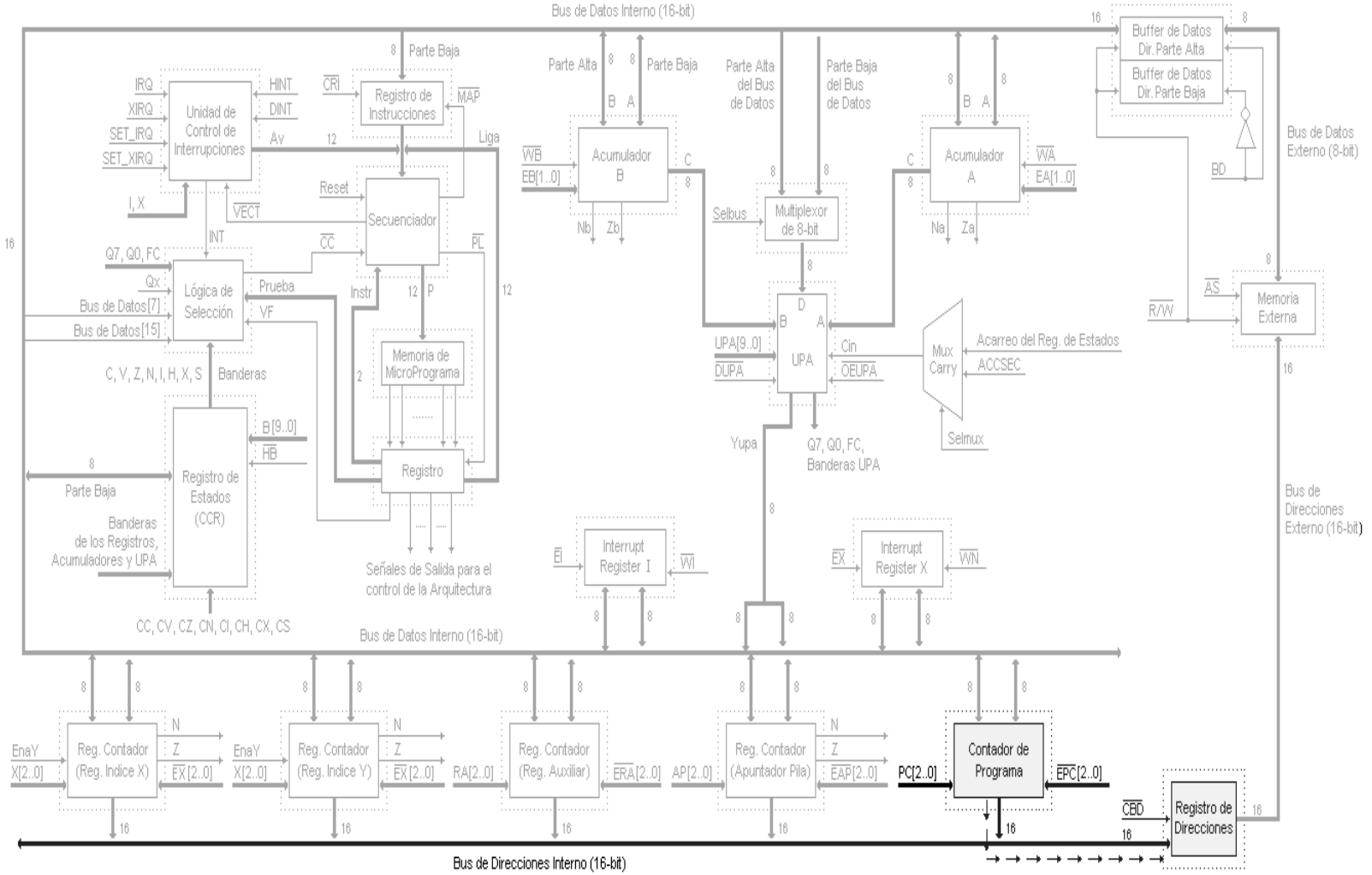


Figura 6.8. Primera fase de ejecución de toda instrucción. Estado TRAER.

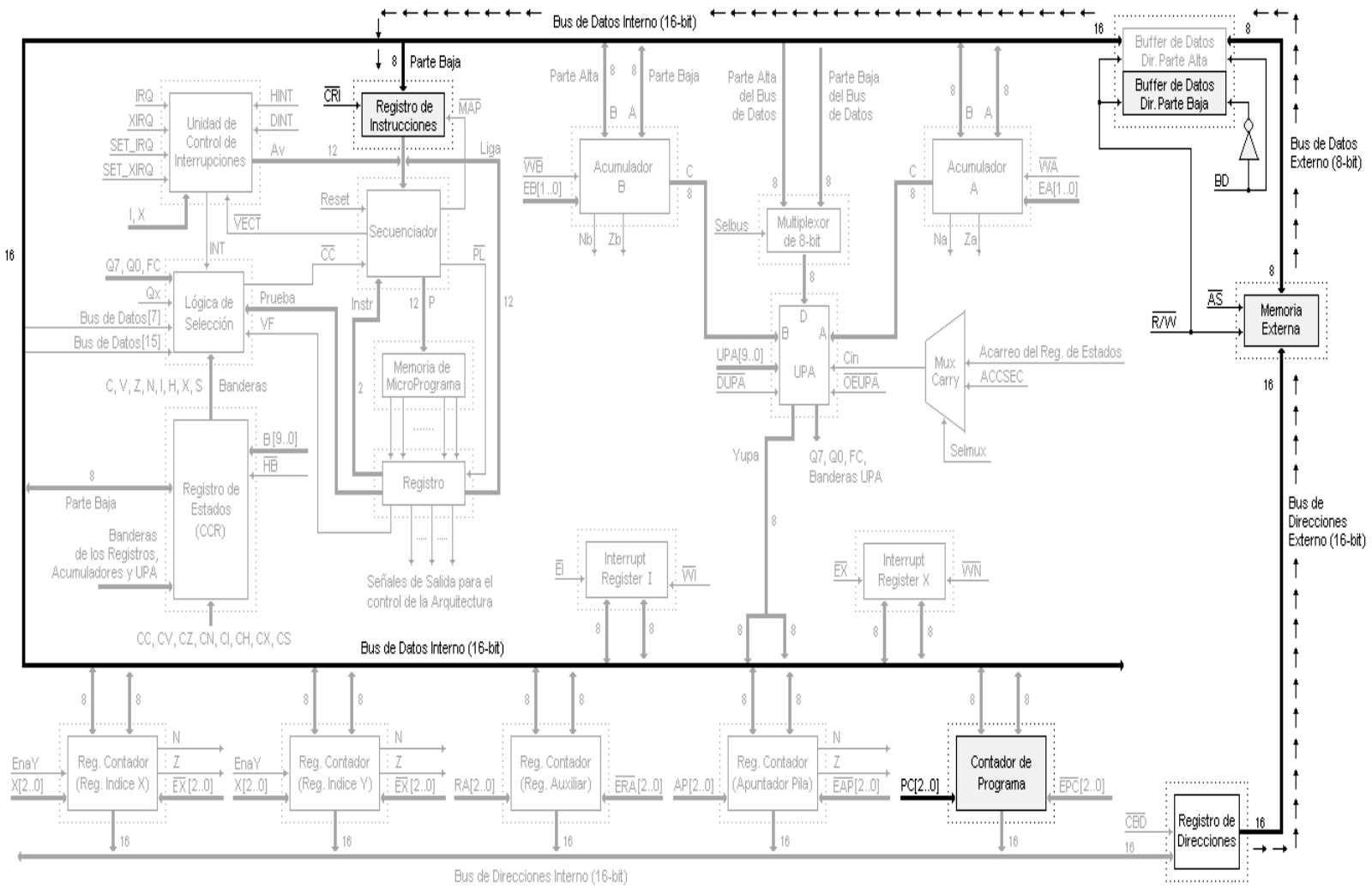


Figura 6.9. Segunda fase de ejecución de toda instrucción. Estado TRAER1.

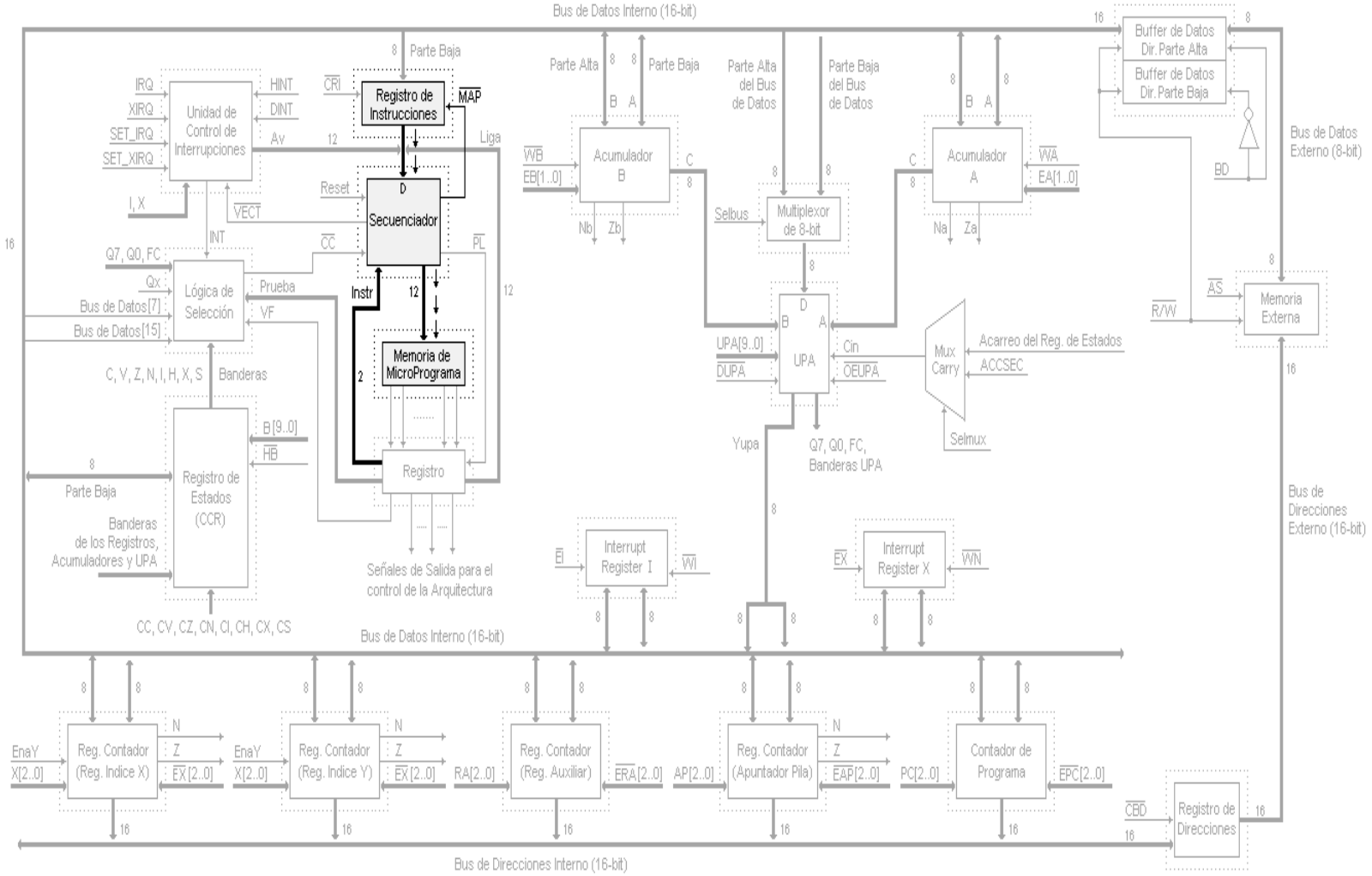


Figura 6.10. Tercera fase de ejecución de toda instrucción. Estado DECOD.

En el estado “INX” comienzan las microinstrucciones necesarias para ejecutar dicha instrucción. En este estado se incrementa el contenido del registro índice X con la activación de la señal X0, es decir, X2X1X0=001. Además, el valor de la señal EnaY debe permanecer en cero, ya que las operaciones que se realizarán serán sobre el registro X. Recuerde que las instrucciones que hacen uso de los registros índices X e Y son idénticas, por ello, ambos registros comparten las mismas líneas de operación y de habilitación. Es por medio de la señal EnaY que se selecciona el registro sobre el que se operará. Más adelante veremos la instrucción INY para ejemplificar el uso de la señal EnaY. El estado “INX” está representado gráficamente en la figura 6.11.

En el estado “INX1” se actualizan los valores de las banderas. Las banderas afectadas se establecen en la especificación de la instrucción. En el caso de la instrucción INX, la bandera de cero (Z) es la única bandera a modificar.

Para actualizar el valor de la bandera de cero en el registro de estados (CCR) debemos habilitar el reloj del flip-flop asociado a esta bandera, es decir, colocar la señal CZ a uno. La habilitación de CZ permitirá cargar un nuevo valor en el registro flip-flop de la bandera Z. La procedencia del nuevo valor de Z se selecciona utilizando las líneas B5, B4 y B3; estas líneas eligen qué bandera de Z cargar en el registro de estados. Por ejemplo, si B5B4B3=011, el nuevo valor de la bandera Z provendrá del registro índice X. La figura 6.12 muestra las actividades realizadas en el estado “INX1”.

En el mismo estado “INX1” se revisa si existen interrupciones. Si ocurrió una interrupción, la señal INT tendrá asignado el valor de uno, y se deberá realizar un salto hacia alguno de los estados de atención a la interrupción. El estado INTI será seleccionado si la línea IRQ fue quien generó la señal de interrupción INT, por el contrario, si la línea XIRQ fue quien generó la interrupción, el estado seleccionado será INTX.

La señal INT y la dirección de salto hacia la rutina de atención a la interrupción son generadas por la unidad de control de interrupciones. La señal INT es utilizada por la lógica de selección para generar la señal CC , y la dirección de salto es utilizada por el secuenciador para saber la dirección de inicio de la microrutina de atención a la interrupción. La instrucción que ejecuta el secuenciador es un salto condicional utilizando la dirección de las interrupciones; recuerde que esta instrucción activa la señal VECT para seleccionar la dirección de salto proveniente de la unidad de control de interrupciones.

Si no existe alguna interrupción entonces se debe saltar al estado “TRAER” para comenzar el ciclo “FETCH” de la siguiente instrucción. Debido a las características del secuenciador, la instrucción de salto condicional con interrupciones no permite ejecutar dos saltos, esto es, si no se realiza el salto hacia la dirección de interrupción entonces el estado siguiente está dado por el estado presente más uno. Por lo tanto, es necesario colocar el estado “INX2” y desde ahí realizar el salto hacia el estado “TRAER”. Con la finalidad de no desperdiciar el estado “INX2” podemos ejecutar en ese mismo estado las microinstrucciones del estado “TRAER” y realizar un salto condicional al estado “TRAER1”. En la figura 6.13 se presentan las actividades ejecutadas en el estado “INX2”.

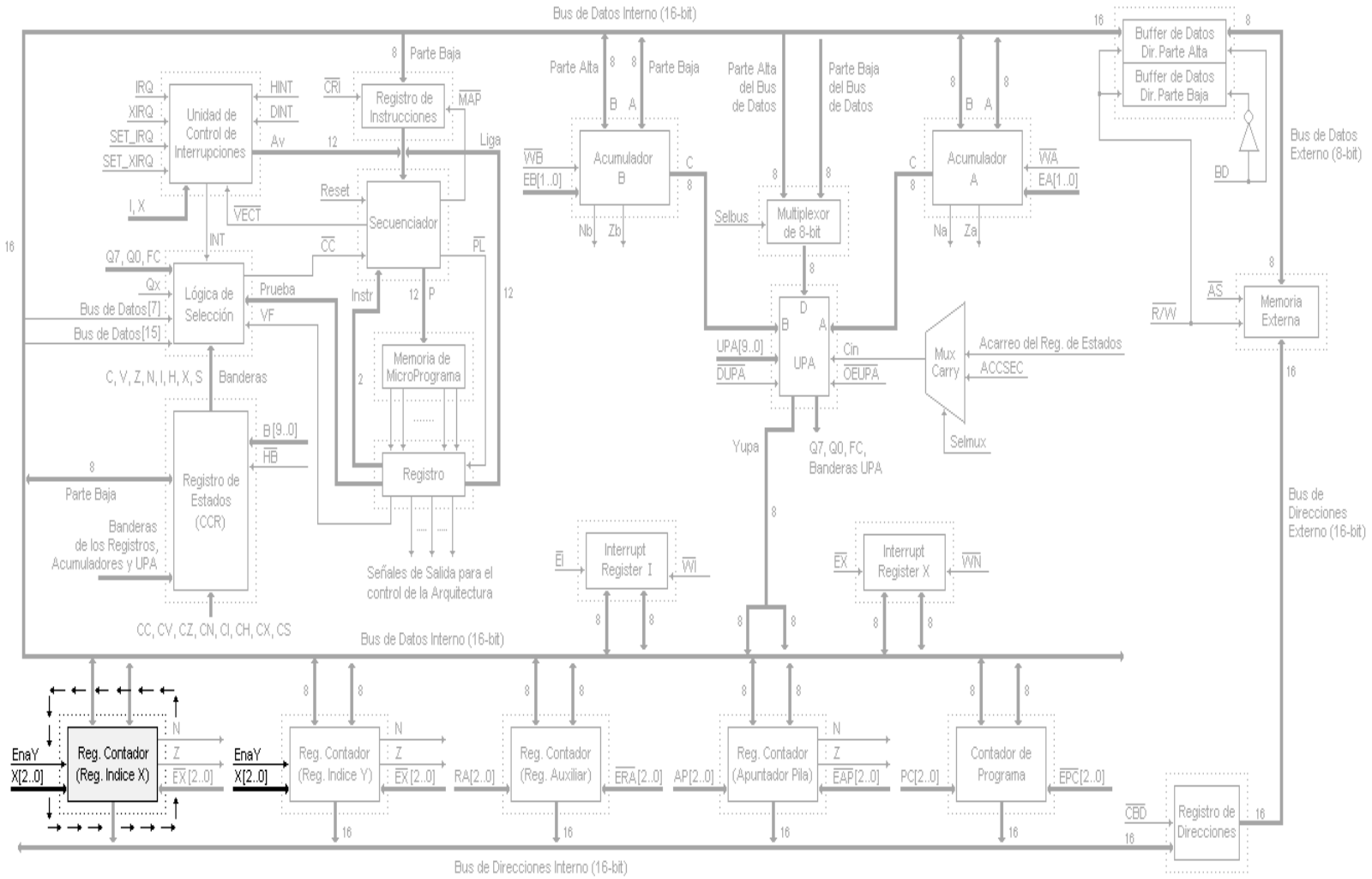


Figura 6.11. Primera fase de ejecución de la instrucción INX. Estado INX.

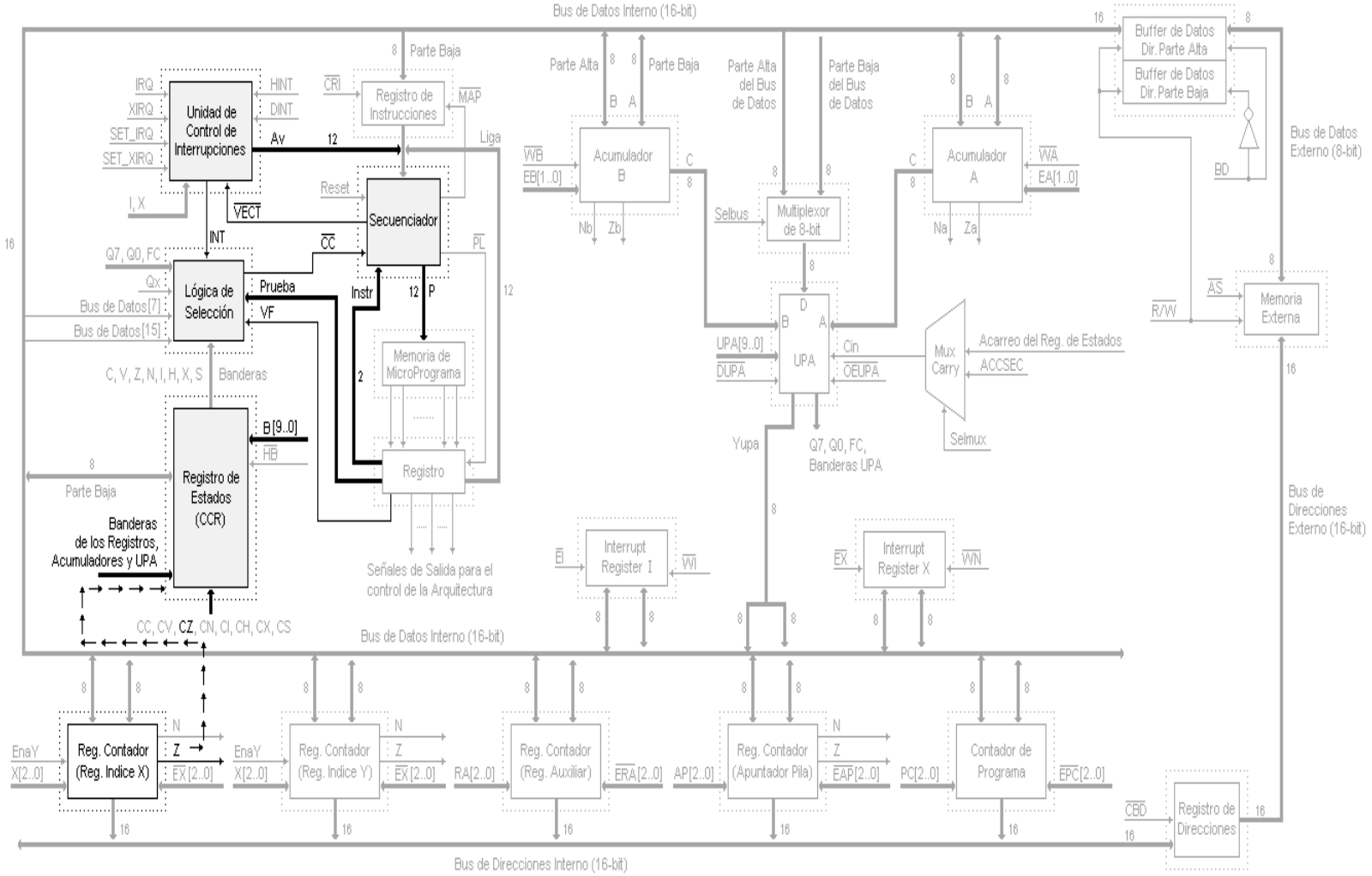


Figura 6.12. Última fase de ejecución de la instrucción INX. Estado INX1.

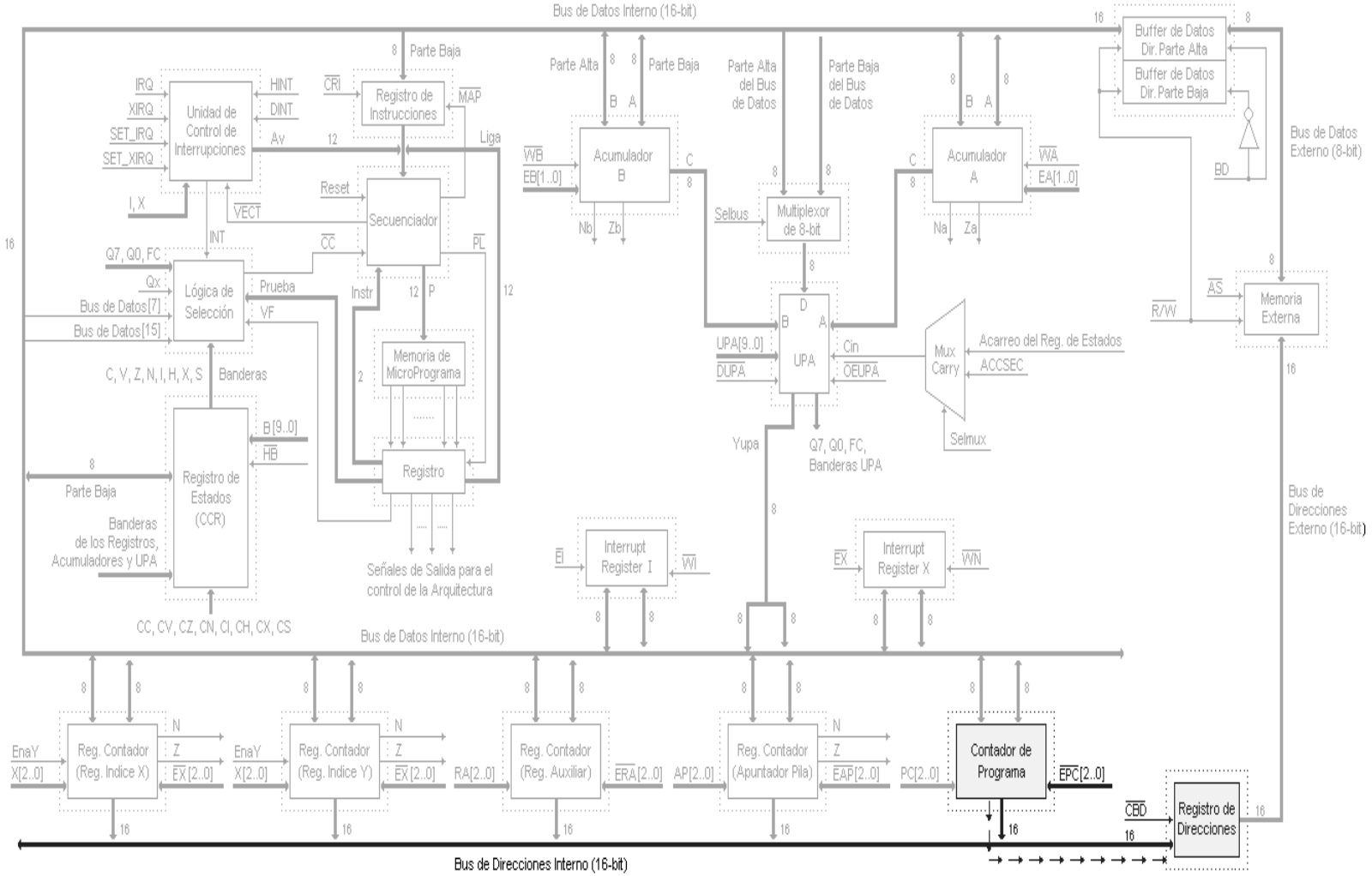


Figura 6.13. Estado INX2. Primera fase de ejecución de toda instrucción. Estado TRAER.

La tabla 6.1 muestra el contenido de la memoria de microprograma para la carta ASM de la figura 6.7. El tamaño de esta memoria es de 4096x100, es decir, 4096 palabras de 100 bits cada una. Por cuestiones de espacio sólo se muestran los bits correspondientes a los siguientes campos: I_1I_0 (la instrucción que ejecuta el secuenciador), Prueba, VF, Liga, y las salidas que intervienen durante la ejecución de las micro-operaciones para la instrucción INX. También por cuestiones de espacio, algunos campos se representan en formato hexadecimal, como Liga y B[9..0]. El resto de los campos se representan en formato binario.

A las variables INT y Q_x les fueron asignados ciertos códigos binarios para poderlas representar dentro del campo de Prueba: INT recibió el código 11111, mientras que Q_x recibió el código 00000. Además, se estableció el valor de Q_x a cero.

Observe que las variables negadas tienen un nivel lógico alto como valor por default. Si estas variables aparecen dentro de un estado en la carta ASM su valor cambia a cero, es decir, se activan. Las variables sin negar tienen un comportamiento inverso, presentan un nivel lógico bajo por default y cuando aparecen en algún estado de las cartas ASM cambian a uno.

DIRECCIÓN	CONTENIDO DE LA MEMORIA															
	Instrucción Secuenciador I1I0	Prueba	VF	Liga	Salidas que controlan la Arquitectura											
					$\overline{EPC}$[2..0]	PC[2..0]	$\overline{CBD}$	$\overline{AS}$	$\overline{R/W}$	$\overline{CRI}$	$\overline{EX}$[2..0]	X[2..0]	EnaY	B[9..0]	CZ	...
0000 0000 1000	00	00000	0	000h	0 1 1	0 0 0	0	1	1	1	1 1 1	0 0 0	0	000h	0	...
0000 0000 1001	00	00000	0	000h	1 1 1	0 0 1	1	0	1	0	1 1 1	0 0 0	0	000h	0	...
0000 0000 1010	10	00000	0	000h	1 1 1	0 0 0	1	1	1	1	1 1 1	0 0 0	0	000h	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0000 1000 0000	00	00000	0	000h	1 1 1	0 0 0	1	1	1	1	1 1 1	0 0 1	0	000h	0	...
0000 1000 0001	11	11111	1	000h	1 1 1	0 0 0	1	1	1	1	1 1 1	0 0 0	0	018h	1	...
0000 1000 0010	01	00000	0	009h	0 1 1	0 0 0	0	1	1	1	1 1 1	0 0 0	0	000h	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Tabla 6.1. Contenido de la memoria de microprograma para la instrucción INX.

Una manera alternativa de construir la unidad de control de la computadora (UCC) es usando los lenguajes de descripción de hardware, como por ejemplo Verilog HDL. Entonces, la codificación de la carta ASM para la instrucción INX quedaría de la siguiente manera.

```
/*********************************************************************************
 *  Módulo: Unidad de Control de la Computadora (UCC)
  *
 *   Este módulo ejecuta las micro-operaciones para la instrucción INX. Observe          *
 *   que sólo se manejaron las señales de control generales y las señales de control     *
 *   necesarias para la ejecución de la instrucción INX (en total 34 señales de salida).  *
 *   Si desea controlar el resto de la arquitectura deberá anexar las señales de salida   *
 *   restantes.                                                                          *
 *********************************************************************************/
module UCC(clk, Reset, CCn, D, MAPn, VECTn, SA, SB, SC);

/***********************************************
 *  Definición de las Señales de Entrada y de Salida *
 ***********************************************/
input             clk, Reset, CCn;
input     [11:0]  D;
output            MAPn, VECTn;
output    [15:0]  SA, SB;
output    [1:0]   SC;
reg       [15:0]  SA, SB;
reg       [1:0]   SC;
reg       [11:0]  EstadoPresente;
reg               MAPn, VECTn;


/***********************************
 *  Generación de las Señales de Salida        *
 ***********************************/
always @ (EstadoPresente) begin
      case (EstadoPresente)

          /*****************************************************************************************
           *  Los vectores SA, SB y SC manejan las señales de salida en el siguiente orden. *
           *                                                                                       *
           *  El vector SA maneja 16 bits pertenecientes a las señales:                            *
           *       Prueba[4:0] VF EPCn[2:0] PC[2:0] CBDn Asn R/Wn CRIn                             *
           *  El vector SB maneja 16 bits pertenecientes a las señales:                            *
           *       EXn[2:0] X[2:0] EnaY B[9:1]                                                     *
           *  El vector SC maneja 2 bits pertenecientes a las señales:                             *
           *       B[0] CZ                                                                         *
          *  Recuerde que la señal de más a la izquierda es la más significativa                  *
           *****************************************************************************************/

          12'h008: // Salidas correspondientes al Estado TRAER
          begin
                SA = 16'b0000000110000111; SB = 16'b1110000000000000; SC = 2'b00;
                MAPn = 1; VECTn = 1;
          end
          12'h009: // Salidas correspondientes al Estado TRAER1
          begin
                SA = 16'b0000001110011010; SB = 16'b1110000000000000; SC = 2'b00;
                MAPn = 1; VECTn = 1;
          end
```

```
            12'h00A: // Salidas correspondientes al Estado DECOD
            begin
                    SA = 16'b0000001110001111; SB = 16'b1110000000000000; SC = 2'b00;
                    MAPn = 0; VECTn = 1;
            end
            12'h080: // Salidas correspondientes al Estado INX
            begin
                    SA = 16'b0000001110001111; SB = 16'b1110000000000000; SC = 2'b00;
                    MAPn = 1; VECTn = 1;
            end
            12'h081: // Salidas correspondientes al Estado INX1
            begin
                    SA = 16'b1111111110001111; SB = 16'b1110000000001100; SC = 2'b01;
                     MAPn = 1; VECTn = 0;
            end
            12'h082: // Salidas correspondientes al Estado INX2 = Estado TRAER
            begin
                    SA = 16'b0000000110000111; SB = 16'b1110000000000000; SC = 2'b00;
                    MAPn = 1; VECTn = 1;
            end
            default:  // Salidas por default
            begin
                    SA = 16'b0000001110001111; SB = 16'b1110000000000000; SC = 2'b00;
                    MAPn = 1; VECTn = 1;
            end
      endcase
end

/*****************************************************
 *  Instrucciones del Secuenciador y Transiciones entre Estados    *
 *****************************************************/
always @ (posedge clk) begin
      if (Reset)
            EstadoPresente = 12'h008;
      else
            case (EstadoPresente)
                    12'h008: // El secuenciador ejecuta la instrucción continúa
                              EstadoPresente = 12'h009;
                    12'h009: // El secuenciador ejecuta la instrucción continúa
                              EstadoPresente = 12'h00A;
                    12'h00A: // El secuenciador ejecuta la instrucción salto de transformación
                              EstadoPresente = D;
                    12'h080: // El secuenciador ejecuta la instrucción continúa
                              EstadoPresente = 12'h081;
                    12'h081: // Se ejecuta la instrucción salto condicional con interrupciones
                              if (CCn)        EstadoPresente = 12'h082;
                              else            EstadoPresente = D;
                    12'h082: // El secuenciador ejecuta la instrucción salto condicional
                              EstadoPresente = 12'h008;
            endcase
end

endmodule
```

En la practica 6 se presenta otra forma de constucción de la arquitectura del 6811 utilizando el leguaje VHDL. Cada una de las instrucciones en ensamblador podría ser programada de la forma anterior, por lo tanto, surge una interrogante:

¿Qué conviene más, construir la Unidad de Control de la Computadora utilizando el secuenciador y la memoria de microprograma, o bien, construirla utilizando los lenguajes de descripción de hardware y dejar que el compilador de tales lenguajes se encargue de generar las funciones lógicas que la describen?

Para contestar esta pregunta se tiene que tomar en consideración el número de elementos lógicos que se utilizan para cada técnica. Para el caso en donde se utiliza el secuenciador, la memoria de microprograma es la que ocupa mayor espacio. Si el repertorio de instrucciones que ejecuta nuestra computadora es muy grande, quizá resulte conveniente seguir el método del secuenciador, ya que si se utiliza el otro método, las ecuaciones lógicas creadas para generar las salidas que controlan a la arquitectura se vuelven cada vez más complejas. Por consiguiente, el segundo método requiere mayor número de compuertas lógicas a medida que aumenta el número de instrucciones en ensamblador.

Es responsabilidad del diseñador decidir qué método utilizar de acuerdo al espacio disponible en el circuito integrado con el que cuente.

6.3.2 INSTRUCCIÓN INY (Acceso Inherente)

Instrucción:	INY
Operación:	IY ⇐(IY) + 1
Código[5]:	18 08
Descripción:	Incrementa el contenido del registro Y en una unidad.
Banderas:	Z=1 si el resultado del incremento es cero, Z=0 en caso contrario.

S	X	H	I	N	Z	V	C
—	—	—	—	—	↕	—	—

Formato: El primer byte del código de la instrucción corresponde al valor hexadecimal 0x18; este byte sirve para indicar que la próxima instrucción a ejecutar utiliza el registro índice Y. El segundo byte del código es el identificador de la instrucción a ejecutar, en este caso es un incremento.

[5] El código de operación de las instrucciones que utilizan el registro Y está precedido por el valor 0x18. Este valor le informa a la arquitectura que la siguiente operación a efectuar es sobre el registro Y.

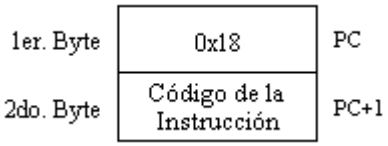


Figura 6.14. Formato de la instrucción INY.

La carta ASM de la figura 6.15 representa el algoritmo que ejecuta la instrucción INY.

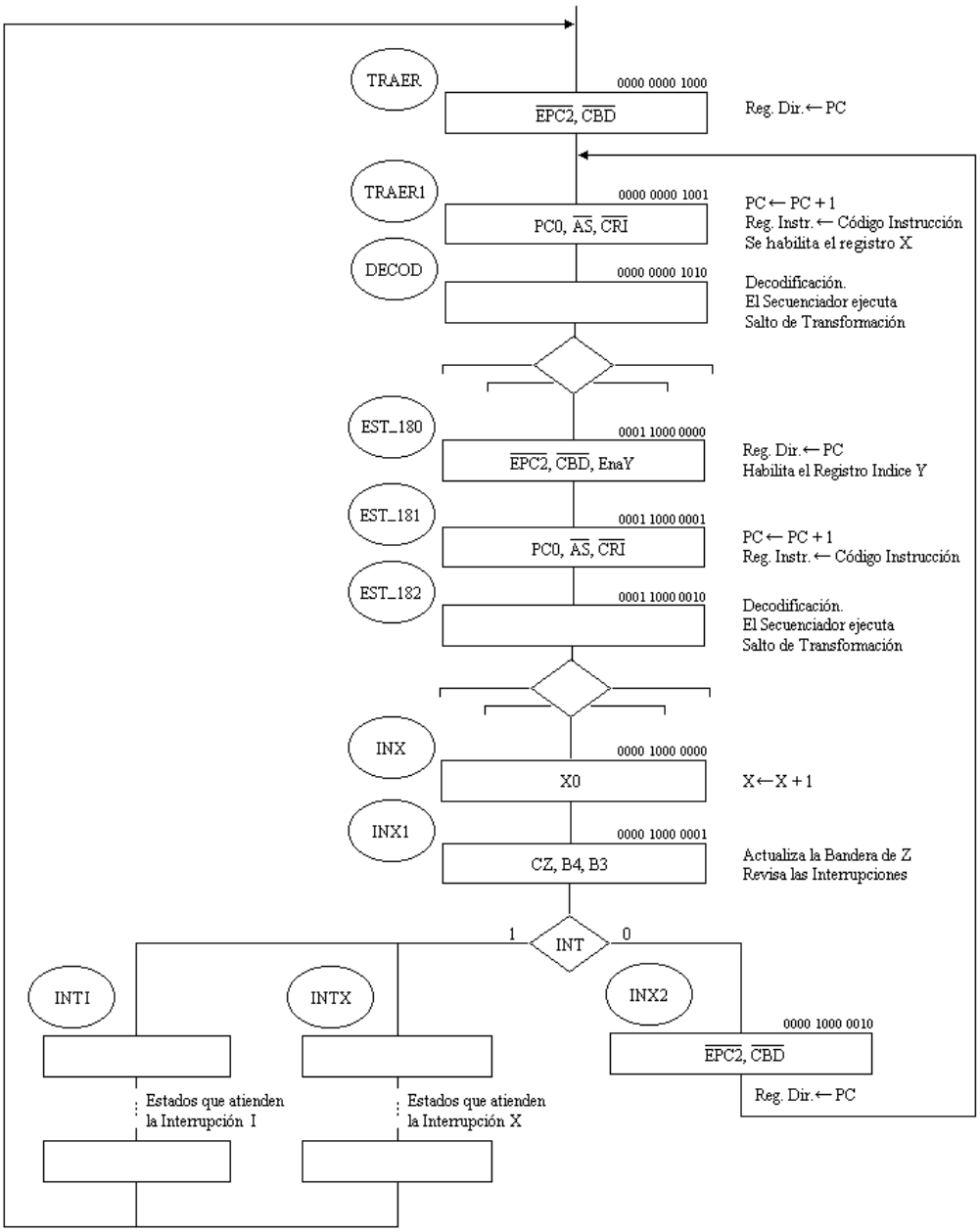


Figura 6.15. Carta ASM para la instrucción INY (acceso inherente).

Los estados “TRAER”, “TRAER1” y “DECOD” son los mismos para todas las instrucciones, en ellos se trae de memoria el código de operación de la próxima instrucción a ejecutar y se decodifica dicha instrucción. Para esta instrucción en particular, en el estado “DECOD”, se decodifica el primer byte de la instrucción INY (0x18) dando como resultado que el secuenciador realice un salto a la dirección 0001 1000 0000 (0x180) en la memoria de microprograma.

En el estado “EST_180” se habilita el registro índice Y, y se comienza un ciclo fetch para traer el segundo byte del código de operación de la instrucción. En este estado, la señal EnaY es la encargada de habilitar al registro Y, el cual permanecerá habilitado hasta que la ejecución de una nueva instrucción comience; durante este tiempo, el registro X estará deshabilitado. En este estado también son habilitadas las señales EPC2 y CBD con el fin de cargar el contenido del registro PC (con la dirección en memoria del segundo byte de la instrucción) en el registro de direcciones.

En el estado “EST_181” se lee de memoria el segundo byte del código de la instrucción y se carga en el registro de instrucción. También es incrementado en una unidad el contenido del registro PC. Por otra parte, en el estado “EST_182” se decodifica el segundo byte del código de la instrucción, este valor le indica al secuenciador la dirección de inicio de las micro-operaciones para la instrucción INY; dicha dirección es la misma que para la instrucción INX (0x080).

En los siguientes estados de la instrucción INY se incrementa el contenido del registro, se actualizan banderas y se revisan las interrupciones, tal y como se realiza para la instrucción INX.

Ahora comparemos a la instrucción INX con la instrucción INY. Observe que los códigos de operación para las instrucciones sobre el registro X no cuentan con un precódigo que especifique explícitamente que las operaciones a ejecutar son sobre el registro X; por lo tanto, es necesario activar las operaciones sobre el registro X, y desactivarlas sobre el registro Y, antes de comenzar una nueva instrucción. De esta manera, si una instrucción utiliza al registro X, las operaciones sobre él estarán activadas; y si una instrucción utiliza al registro Y, entonces, el precódigo se encargará de activar al registro Y.

Como se mencionó con anterioridad, la activación del registro X y la desactivación del registro Y será al inicio de cada instrucción, específicamente en el estado ‘TRAER1’ del ciclo fetch, el cual es ejecutado por todas las instrucciones antes de empezar la ejecución de las micro-operaciones propias de la instrucción. Note que no hay conflictos para las instrucciones que utilizan el registro Y porque el precódigo de estas instrucciones será suficiente para habilitar al registro índice Y y desactivar al registro X.

6.3.3 INSTRUCCIÓN XGDX (Acceso Inherente)

Instrucción: XGDX
Operación: (IX) ⇔ (ACCD)
Código: 8F
Descripción: El contenido del acumulador D es transferido al registro índice X, y el contenido del registro índice X es transferido al doble acumulador D.
Banderas: Ninguna bandera es afectada.

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

A continuación se presenta la carta ASM que ejecuta esta instrucción.

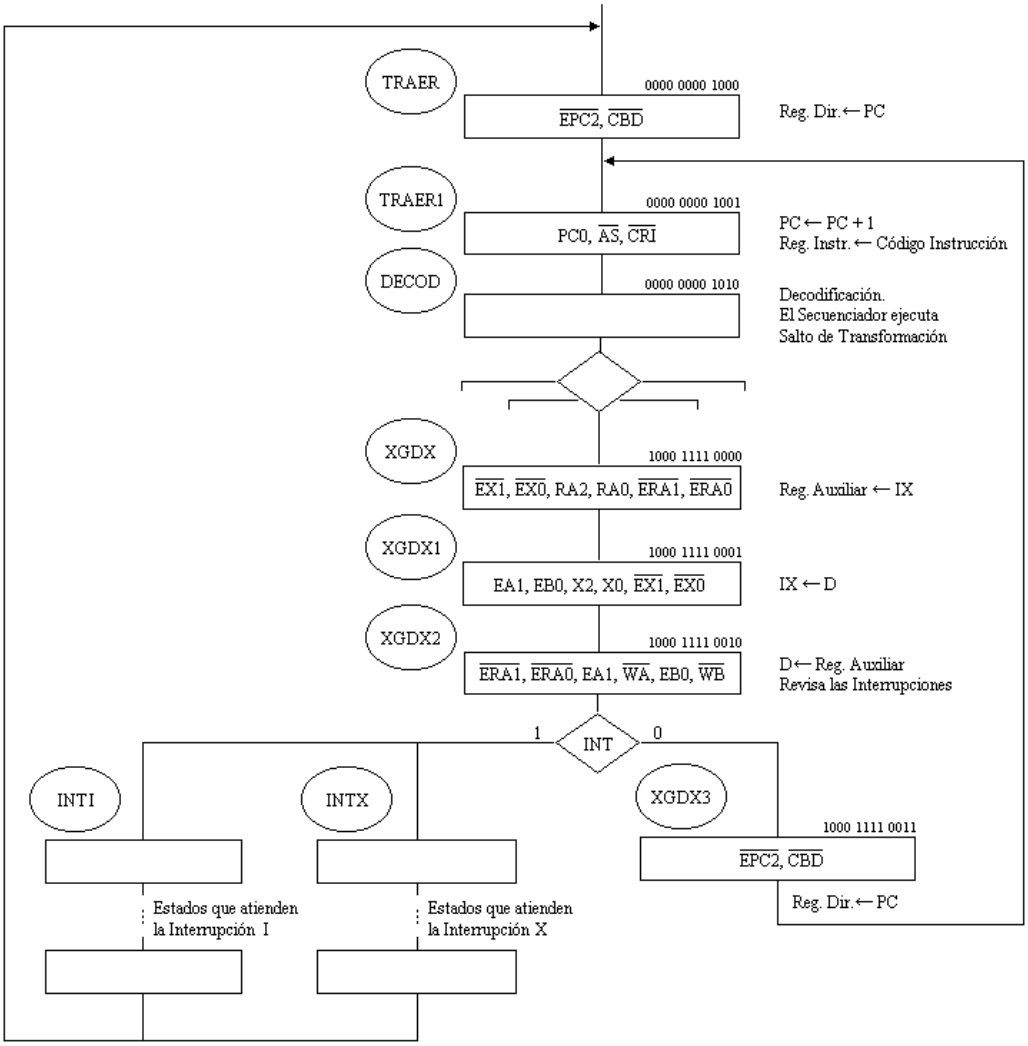


Figura 6.16. Carta ASM para la instrucción XGDX (acceso inherente).

6.3.4 INSTRUCCIÓN LDAB (Acceso Inmediato)

Instrucción:	LDAB #DATO
Operación:	ACCB ⇐(Memoria)
Código:	C6
Descripción:	Carga en el acumulador B, un dato inmediato de 8 bits contenido en memoria.
Banderas:	N=1 si el MSB[6] del resultado está encendido, N=0 en caso contrario. Z=1 si el resultado en el registro es cero, Z=0 en caso contrario. V se coloca a cero.

[6] MSB = Bit más significativo.

S	X	H	I	N	Z	V	C
—	—	—	—	↕	↕	0	—

A continuación se presenta la carta ASM que ejecuta esta instrucción.

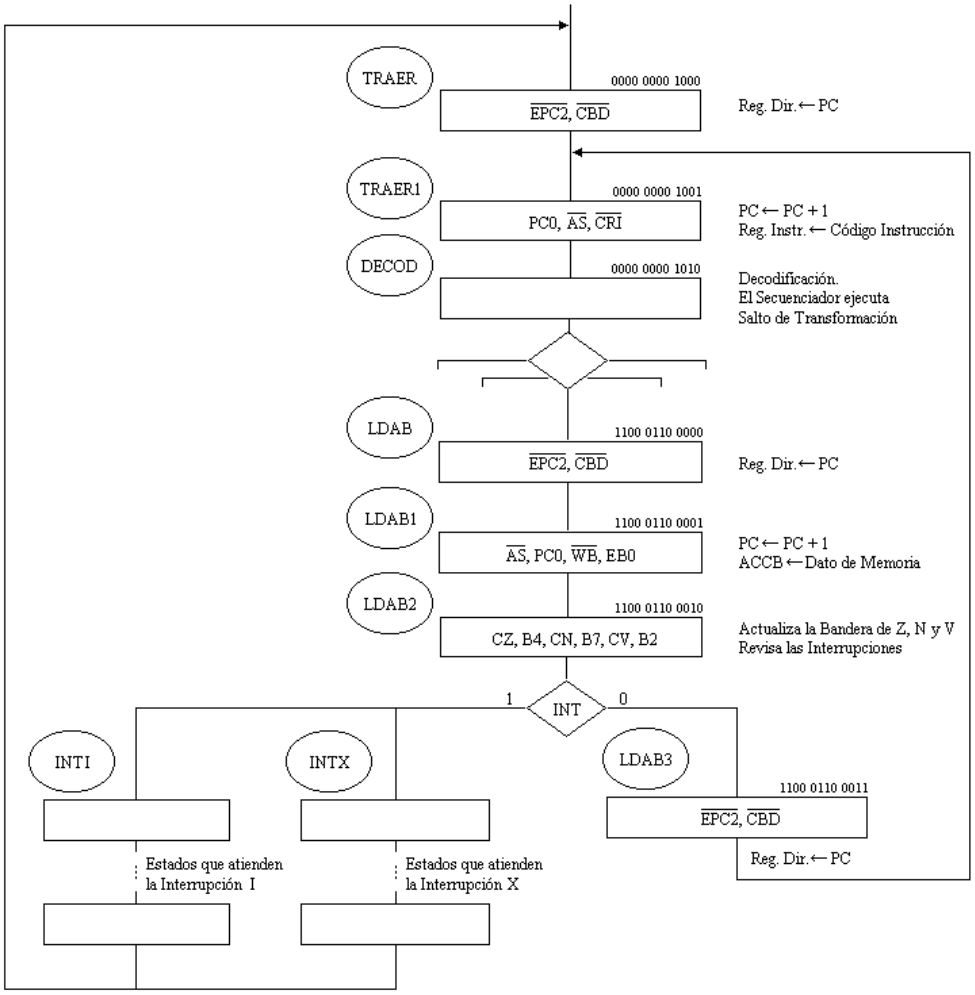


Figura 6.17. Carta ASM para la instrucción LDAB (acceso inmediato).

6.3.5 INSTRUCCIÓN LDAA (Acceso Inmediato)

Instrucción: LDAA #DATO
Operación: ACCA ⇐(Memoria)
Código: 86
Descripción: Carga en el acumulador A, un dato inmediato de 8 bits contenido en memoria.
Banderas: N=1 si el MSB del resultado está encendido, N=0 en caso contrario.
Z=1 si el resultado en el registro es cero, Z=0 en caso contrario.

V se coloca a cero.

S	X	H	I	N	Z	V	C
—	—	—	—	↕	↕	0	—

A continuación se presenta la carta ASM que ejecuta esta instrucción.

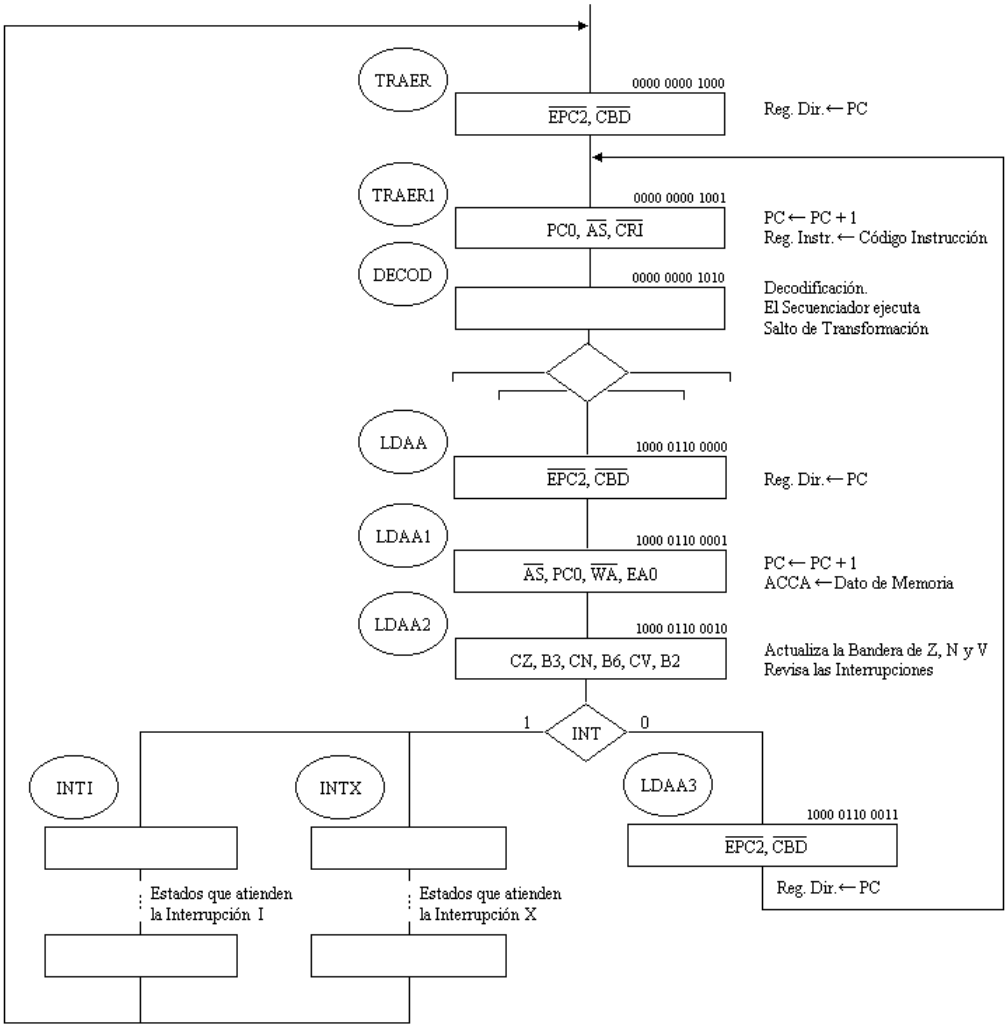


Figura 6.18. Carta ASM para la instrucción LDAA (acceso inmediato).

6.3.6 INSTRUCCIÓN SUBA (Acceso Extendido)

Instrucción: SUBA Dirección_16_Bits
Operación: ACCA ⇐(ACCA) – (Memoria)
Código: B0

Descripción: Resta al contenido del acumulador A el contenido de la Memoria. El resultado es almacenado nuevamente en el acumulador A.

Banderas: N=1 si el MSB del resultado de la resta es igual a uno, N=0 en caso contrario.
Z=1 si el resultado de la resta es cero, Z=0 en caso contrario.
V= ACCA7 • MEM7 • RES7 + ACCA7 • MEM7 • RES7
C= ACCA7 • MEM7 + MEM7 • RES7 + RES7 • ACCA7

donde, ACCA7 = Bit más significativo del dato contenido en el acumulador A.
MEM7 = Bit más significativo del dato contenido en memoria.
RES7 = Bit más significativo del resultado de la resta.

S	X	H	I	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

Formato: El primer byte del código de operación corresponde al valor hexadecimal 0xB0; el segundo byte es la parte alta de una localidad de memoria de 16 bits; y el tercer byte es la parte baja de dicha localidad En esta localidad de memoria se encuentra el dato que será restado al contenido del acumulador A.

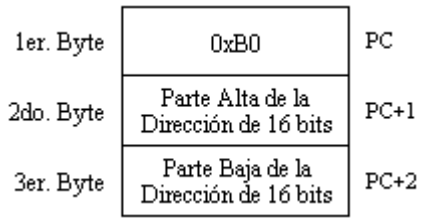


Figura 6.19. Formato de la instrucción SUBA.

En la figura 6.20 se presenta la carta ASM que ejecuta esta instrucción.

En los primeros estados de la carta ASM se realiza el ciclo fetch y se decodifica la instrucción SUBA. En los estados “SUBA” a “SUBA4” se obtiene la dirección en memoria del dato a restar, y se lee el dato contenido en dicha dirección.

En el estado “SUBA5” se efectúa la resta de los operandos, por lo tanto, se le indica a la unidad de procesos aritméticos (UPA) que reste al dato de la entrada A el dato de la entrada D, es decir, que reste al contenido del acumulador A el dato de la memoria. Recuerde que la operación de resta en la UPA se define de la siguiente manera: $S - R - C_{in}$; por lo tanto, es necesario colocar el valor de C_{in} a cero para que el resultado de la resta no sea afectado por un valor de C_{in} indeseado. Para ello, se activa la salida ACCSEC, y mediante la activación de señal Selmux se selecciona a ACCSEC como el acarreo de entrada a la UPA.

En el último estado, “SUBA6”, se actualizan los valores de las banderas de negativo (N), cero (Z), sobreflujo (V) y acarreo (C) de acuerdo a los valores calculados por la UPA, y se verifica si existe alguna interrupción.

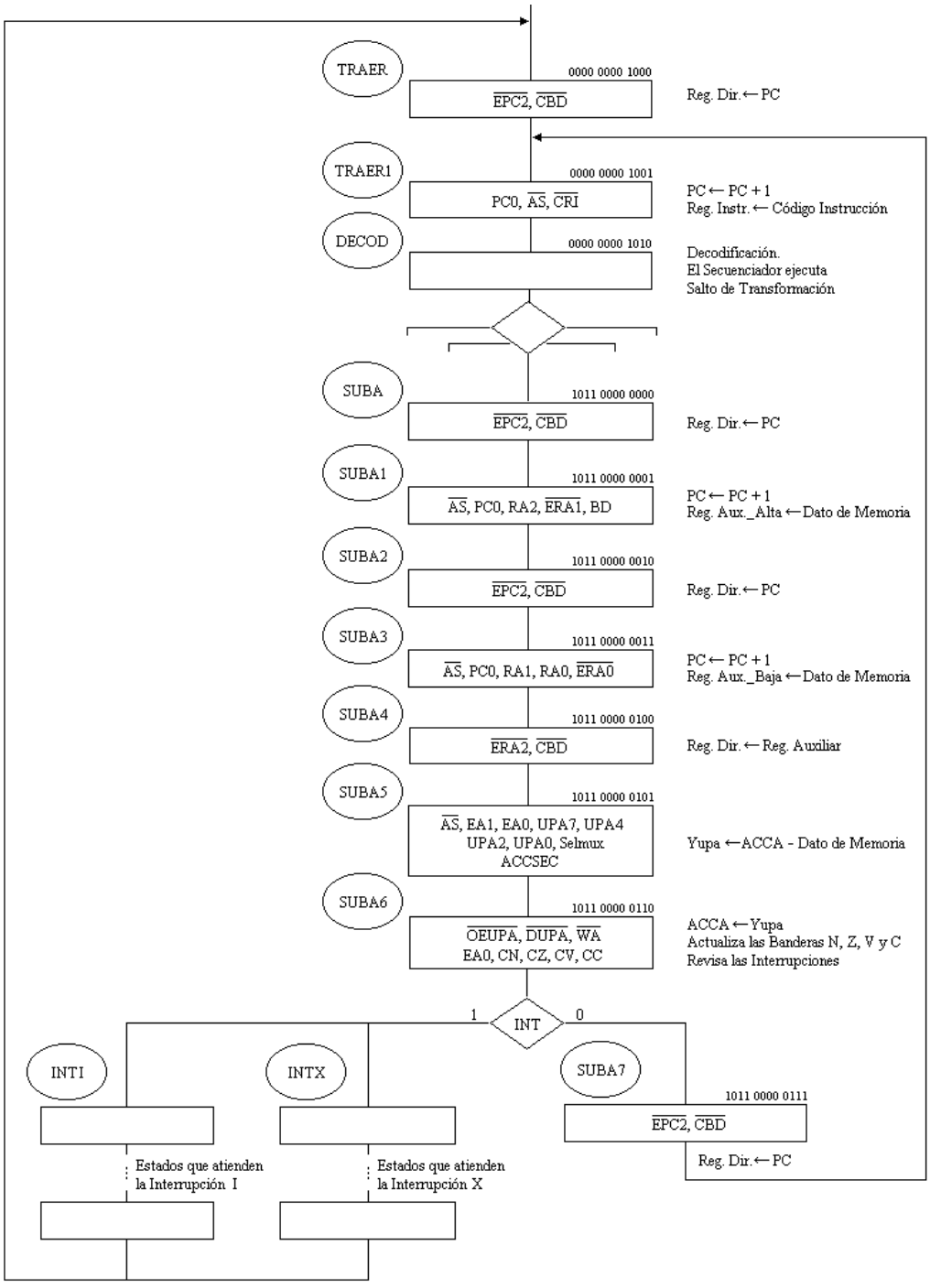


Figura 6.20. Carta ASM para la instrucción SUBA (acceso extendido).

6.3.7 INSTRUCCIÓN BRA (Acceso Relativo)

Instrucción: BRA Desplazamiento
Operación: PC ⇐(PC) + 2 + Desplazamiento
Código: 20
Descripción: Salto incondicional a la dirección: PC + 2 + Desplazamiento. Donde el Desplazamiento es un número en complemento a dos.

Banderas: Ninguna bandera es afectada.

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

Formato: El formato de esta instrucción es el siguiente. El primer byte corresponde al código de operación de la instrucción, es decir, 0x20. El segundo byte corresponde al desplazamiento en complemento a dos.

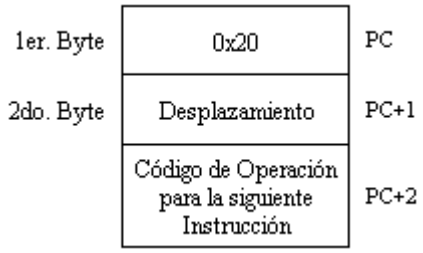


Figura 6.21. Formato de la instrucción BRA.

La dirección de salto se obtiene sumando al contenido del registro PC el valor del desplazamiento. Antes de calcular esta dirección de salto es necesario que PC apunte a la dirección de la siguiente instrucción en memoria, es decir, PC+2. Una vez que PC apunta a la dirección correcta podremos sumarle el desplazamiento.

La suma de PC y el desplazamiento se hace en dos pasos, ya que la UPA sólo puede efectuar operaciones de 8 bits y PC es de 16 bits. En el primer paso se suma a la parte baja del PC el valor del desplazamiento, y en el segundo paso, el valor del acarreo resultante de la primera operación se suma o se resta a la parte alta del PC, según el signo del desplazamiento.

Por ejemplo, supongamos que PC+2 es igual a 0000 1000 1111 1100 y el desplazamiento es igual a 0000 0100 (un desplazamiento positivo), por lo tanto, el nuevo valor de PC será el siguiente.

```
    0 0 0 0  1 0 0 0  1 1 1 1  1 1 0 0
+   0 0 0 0  0 0 0 0  0 0 0 0  0 1 0 0
-------------------------------------
    0 0 0 0  1 0 0 1  0 0 0 0  0 0 0 0
```

Esta suma se puede hacer en dos partes:

$PC_{Baja} \leftarrow PC_{Baja}$ + Desplazamiento
$PC_{Alta} \leftarrow PC_{Alta}$ + Acarreo (de la suma anterior)

Ahora suponga un desplazamiento negativo. La suma es la siguiente.

	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0
+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0

Esta suma también se puede hacer en dos partes:

$PC_{Baja} \leftarrow PC_{Baja}$ + Desplazamiento
$PC_{Alta} \leftarrow PC_{Alta} + (-1)$ + Acarreo (de la suma anterior)

Si el acarreo es cero:

$PC_{Alta} \leftarrow PC_{Alta} + (-1) + 0 = PC_{Alta} -$ Acarreo

Si el acarreo es uno:

$PC_{Alta} \leftarrow PC_{Alta} + (-1) + 1 = PC_{Alta} + 0 = PC_{Alta} -$ Acarreo

Entonces, para ambos casos:

$PC_{Alta} \leftarrow PC_{Alta} -$ Acarreo

En la figura 6.22 se presenta la carta ASM que ejecuta esta instrucción.

Los primeros tres estados de la carta ASM sirven para traer el código de operación de la instrucción y para decodificarla.

En el estado “BRA” guardamos una copia del registro de estados en la parte baja del registro auxiliar, con el fin de no alterar las banderas cuando se realice la suma de PC más el desplazamiento. La activación de las señales HB, ERA0, RA1 y RA0 permiten guardar el contenido del registro de estados en la parte baja del registro auxiliar. En este mismo estado se guarda en el registro de direcciones la dirección en memoria del desplazamiento, la cual está almacenada en PC. Esto se realiza mediante la activación de las señales EPC2 y CBD.

En el estado “BRA1” se guarda el desplazamiento leído de memoria en el registro Q de la UPA, y se incrementa en una unidad el contenido de PC. En este momento PC apunta a la dirección en memoria de la siguiente instrucción a ejecutar, así que ya se tiene el valor correcto de PC para calcular el salto. Recuerde que en el estado “TRAER1” se incrementó por primera vez el PC. También observe que la única forma de guardar el valor del desplazamiento en el registro Q, sin alterar el contenido de los acumuladores ni el valor del desplazamiento, es a través de la entrada D de la UPA y aplicando la función lógica OR entre el desplazamiento y el valor de cero.

En el estado “BRA2” se suma a la parte baja del PC el valor del desplazamiento y el resultado se guarda en el registro Y_{upa} de la UPA (el contenido del registro Q no se modifica). Note que el acarreo de entrada a la UPA para esta operación es ACCSEC=0, el cual se selecciona por medio de la línea Selmux.

En el estado “BRA3” se guarda el resultado de la suma en la parte baja de PC y se almacena en el registro de estados el valor de la bandera de acarreo obtenido en la suma. Tenga presente que en este estado la señal DUPA es habilitada para no modificar el resultado calculado en el estado anterior. Por otra parte, la señal OEUPA habilita el bus de salida de la UPA para que el resultado pueda ser cargado en la parte baja de PC. En este estado también se pregunta por el valor del bit más significativo del desplazamiento, ya que este valor determina si se suma ó se resta, el valor del acarreo a la parte alta de PC.

Si el bit más significativo del desplazamiento es cero entonces se transita al estado “BRA4”. En este estado se suma a la parte alta de PC el valor del acarreo del registro de estados. Recuerde que este acarreo se obtuvo cuando fueron sumados la parte baja de PC y el desplazamiento. Por el contrario, si el bit más significativo del desplazamiento es uno, entonces, se transita al estado “BRA8”. En “BRA8” se resta a la parte alta de PC el valor negado del acarreo guardado en el registro de estados.

En el estado “BRA5” se guarda el resultado de la segunda suma ó de la resta en la parte alta de PC, para ello se activan las señales PC2, OEUPA , DUPA y EPC1. En el estado “BRA6” la activación de las señales ERA0 , CC y B0, permiten restaurar el valor original de la bandera de acarreo en el registro de estados. También en este estado se revisa la existencia de interrupciones, si existe alguna interrupción, entonces, se salta al estado de atención a la interrupción, si no, se salta al estado “BRA7”.

En “BRA7” se ejecuta la primera instrucción del ciclo Fetch y se salta al estado “TRAER1”.

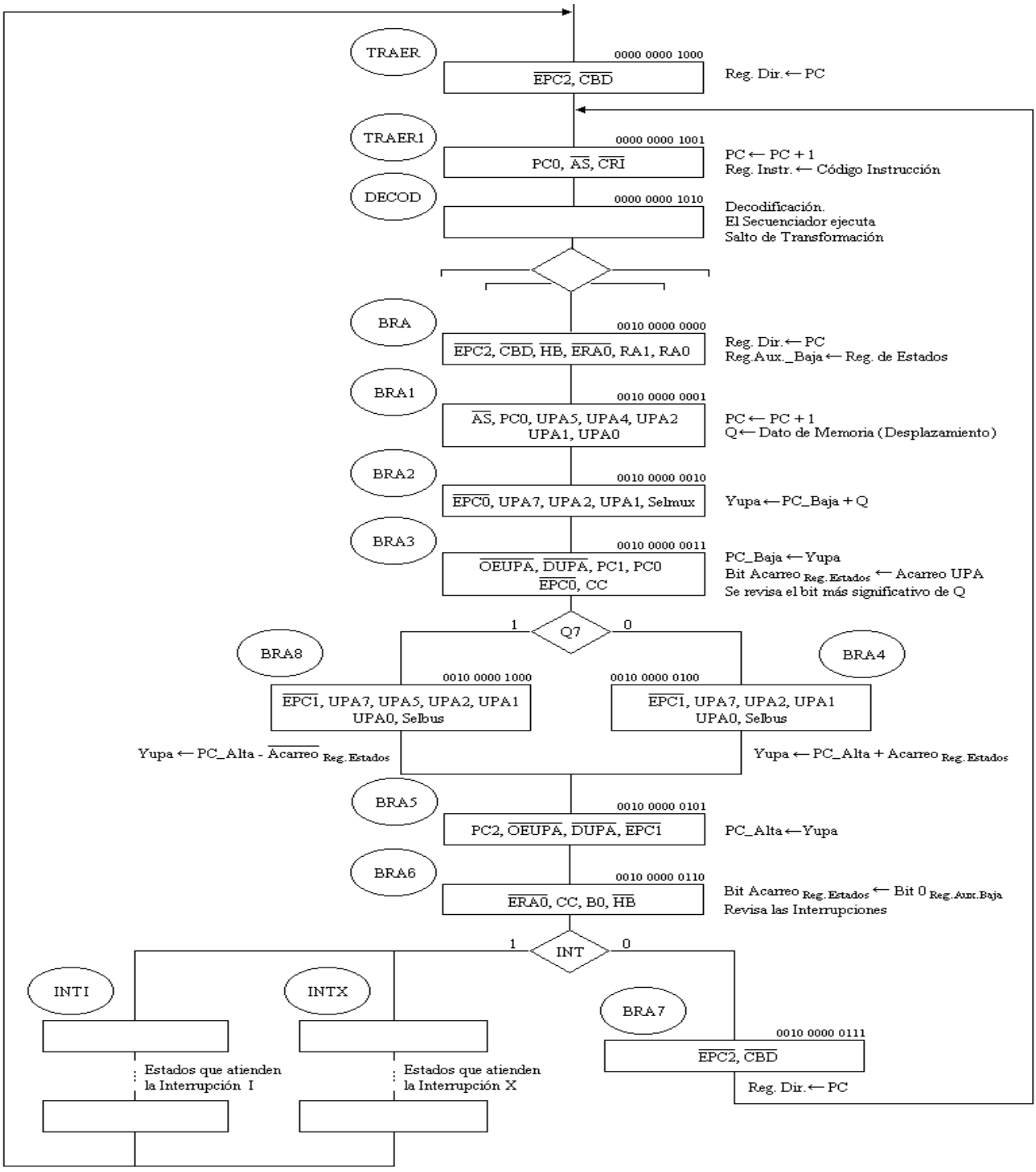


Figura 6.22. Carta ASM para la instrucción BRA (acceso relativo).

6.3.8 INSTRUCCIÓN BEQ (Acceso Relativo)

Instrucción: BEQ Desplazamiento
Operación: Si Z=1, PC ⇐(PC) + 2 + Desplazamiento
Código: 27
Descripción: Salto condicional. Si Z=1 entonces PC ⇐(PC) + 2 + Desplazamiento, si Z=0 entonces PC ⇐(PC) + 2.
Banderas: Ninguna bandera es afectada.

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

Formato: El formato de esta instrucción es el siguiente. El primer byte corresponde al código de operación de la instrucción, es decir, 0x27. El segundo byte corresponde al desplazamiento en complemento a dos.

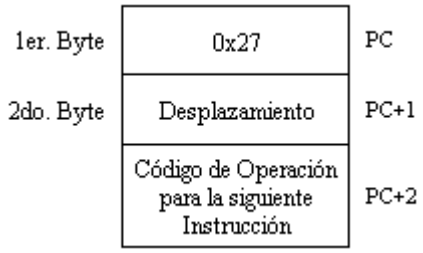


Figura 6.23. Formato de la instrucción BEQ.

En la figura 6.24 se presenta la carta ASM que ejecuta esta instrucción.

Los primeros tres estados de la carta ASM sirven para traer el código de operación de la instrucción y para decodificarla.

En el estado “BEQ” guardamos una copia del registro de estados en la parte baja del registro auxiliar, para ello, son activadas las señales HB, ERA0, RA1 y RA0. También se guarda en el registro de direcciones la dirección en memoria del desplazamiento, la cual está almacenada en PC. Esto se realiza mediante la activación de las señales EPC2 y CBD. Por último, se revisa el valor de la bandera de cero (Z), ya que de este valor depende si se efectúa o no el salto.

Si la bandera Z vale cero, entonces el estado siguiente es “BEQ1”. En “BEQ1” se incrementa el PC para apuntar a la dirección en memoria de la siguiente instrucción, y se revisa la existencia de interrupciones.

Si la bandera Z vale uno, entonces se trae el valor del desplazamiento, y se calcula la dirección de salto sumando al PC el valor de dicho desplazamiento, es decir, se comienzan a ejecutar las mismas

micro-operaciones que en la instrucción BRA. Por lo tanto, se podría saltar directamente al estado de inicio de la instrucción BRA; sin embargo, para aprovechar eficientemente el uso de los estados, en el estado "BEQ", además de preguntar por la bandera Z, también se ejecutan las micro-operaciones del estado "BRA", de manera que si Z es igual a uno, entonces se salta al estado "BRA1". Recuerde que menos estados en las cartas ASM se traduce en computadoras más rápidas.

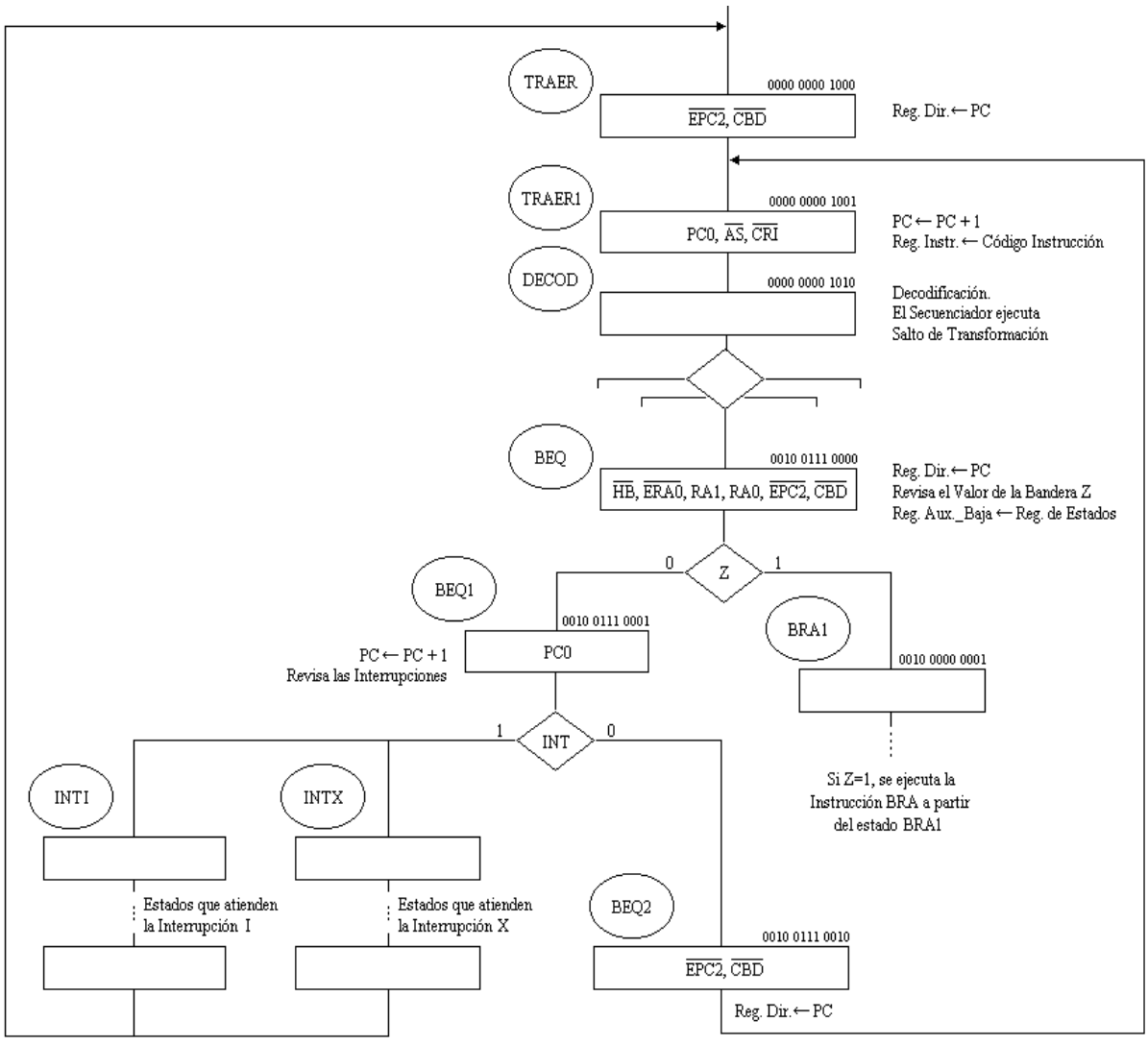


Figura 6.24. Carta ASM para la instrucción BEQ (acceso relativo).

6.3.9 INSTRUCCIÓN JSR (Acceso Extendido)

Instrucción: JSR Dirección_Subrutina_16bits

Operación:

PC ⇐(PC) + 3	
⇓ (PC $_{Baja}$)	Pila ⇐Parte baja de la dirección de regreso
AP ⇐(AP) - 1	
⇓(PC $_{Alta}$)	Pila ⇐Parte alta de la dirección de regreso
AP ⇐(AP) - 1	
PC ⇐Dirección	PC ⇐Dirección de inicio de la subrutina

Código: BD

Descripción: Salto a subrutina. Ocurre un salto hacia la instrucción contenida en la dirección de 16 bits. Antes de efectuar el salto, se guarda en la pila, la dirección de regreso de la subrutina.

Banderas: Ninguna bandera es afectada.

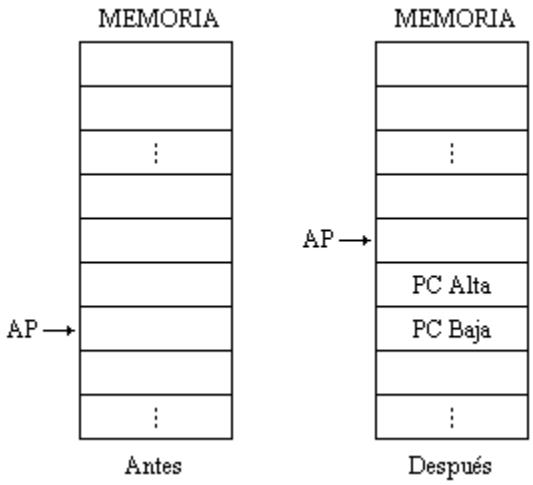
S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

Formato: El primer byte corresponde al código de operación de la instrucción, 0xBD, y los dos bytes siguientes a la dirección de 16 bits donde comienza la subrutina.

1er. Byte	0xBD	PC
2do. Byte	Parte Alta de la Dirección de la Subrutina	PC+1
3er. Byte	Parte Baja de la Dirección de la Subrutina	PC+2
	Código de Operación para la siguiente Instrucción	PC+3

Figura 6.25. Formato de la instrucción JSR.

El contenido de la pila después de ejecutar está instrucción queda de la siguiente manera.



A continuación se presenta la carta ASM que ejecuta esta instrucción.

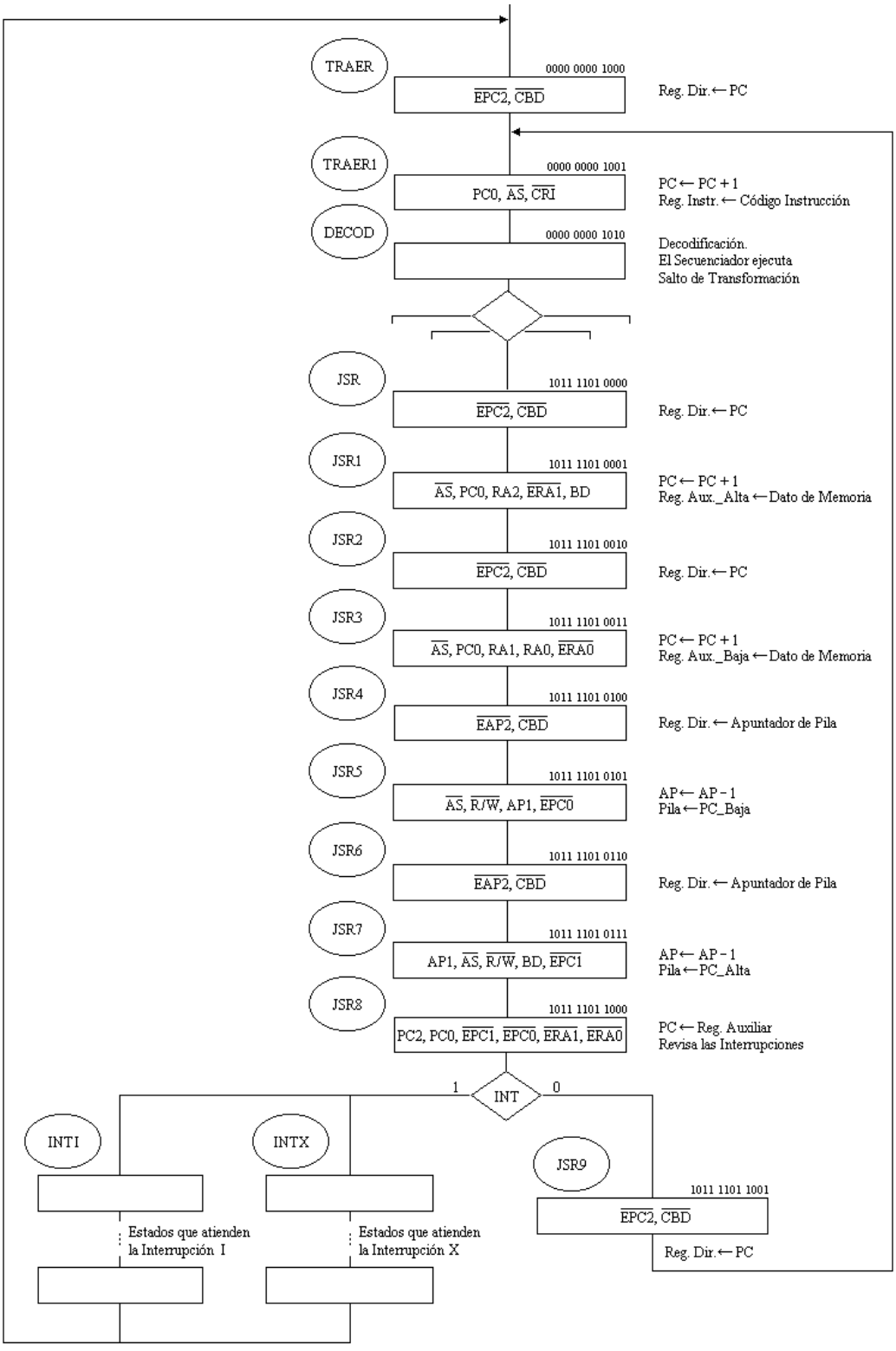


Figura 6.26. Carta ASM para la instrucción JSR (acceso extendido).

6.3.10 INSTRUCCIÓN RTS (Acceso Inherente)

Instrucción: RTS

Operación: AP ⇐(AP) + 1

⇑ (PC $_{Alta}$) Recupera la parte alta de la dirección de regreso

AP ⇐(AP) + 1

⇑(PC $_{Baja}$) Recupera la parte baja de la dirección de regreso

Código: 39

Descripción: Regreso de Subrutina. El registro PC es cargado con la dirección de regreso de la subrutina, la cual se encuentra guardada en la pila.

Banderas: Ninguna bandera es afectada.

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

Formato: Esta instrucción está compuesta por un sólo byte que corresponde al código de operación de la instrucción, 0x39.

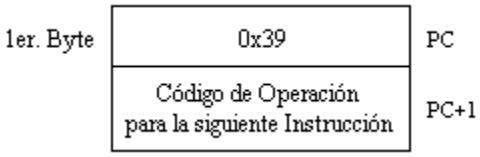
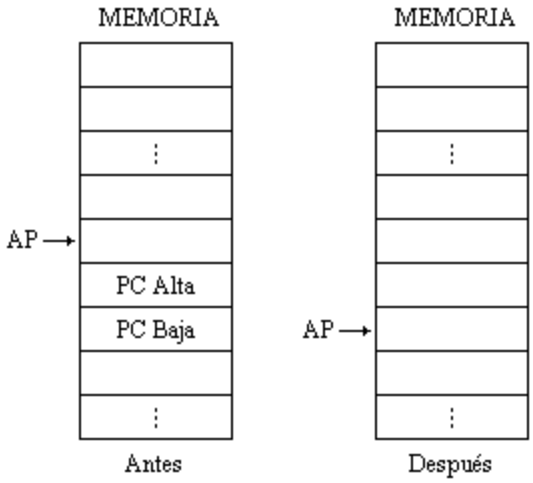


Figura 6.27. Formato de la instrucción RTS.

El contenido de la pila después de ejecutar está instrucción queda de la siguiente manera.



A continuación se presenta la carta ASM que ejecuta esta instrucción.

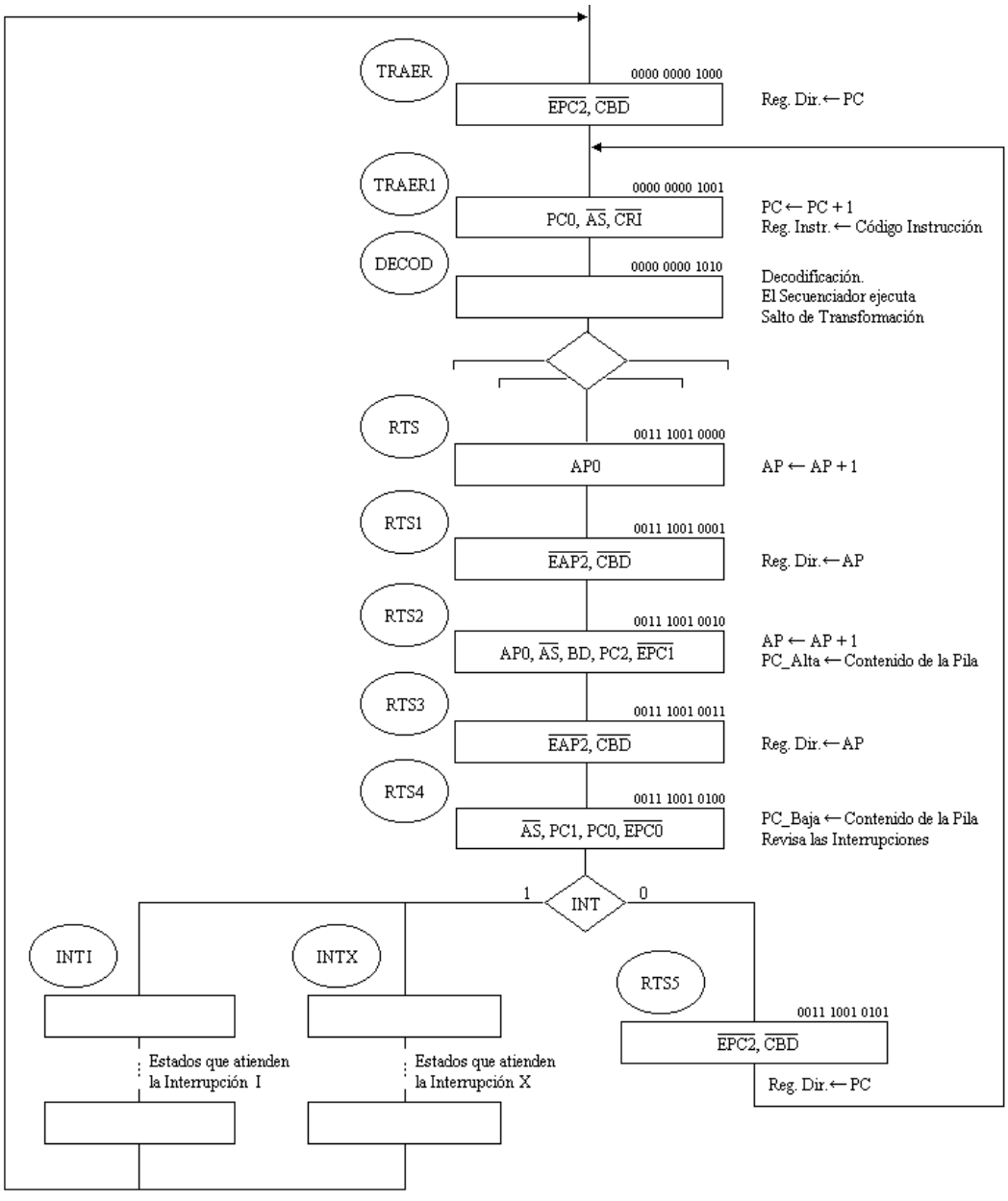


Figura 6.28. Carta ASM para la instrucción RTS (acceso inherente).

6.3.11 ATENCIÓN A INTERRUPCIONES

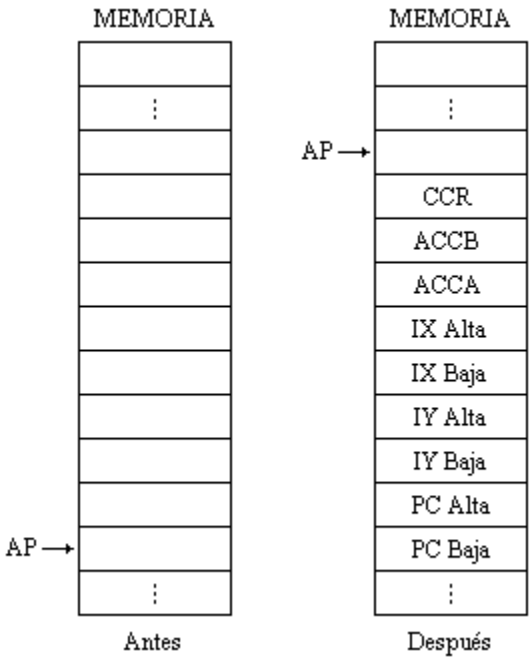
Operación:
⇓(PC $_{Baja}$), AP ⇐(AP) - 1
⇓(PC $_{Alta}$), AP ⇐(AP) - 1
⇓(IY $_{Baja}$), AP ⇐(AP) - 1
⇓(IY $_{Alta}$), AP ⇐(AP) - 1
⇓(IX $_{Baja}$), AP ⇐(AP) - 1
⇓(IX $_{Alta}$), AP ⇐(AP) - 1
⇓(ACCA), AP ⇐(AP) - 1
⇓(ACCB), AP ⇐(AP) - 1
⇓(CCR), AP ⇐(AP) - 1

Descripción: Antes de atender una petición de interrupción se debe guardar el contenido del registro PC, del registro IX, del registro IY, del acumulador A, del acumulador B y del registro de estados (CCR). Una vez guardado el status de la arquitectura, entonces cargamos el PC con la dirección de inicio del manejador que atiende a la interrupción.

Banderas: Ninguna bandera es afectada.

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

El contenido de la pila después de ejecutar está instrucción queda de la siguiente manera.



A continuación se presenta la carta ASM para las interrupciones.

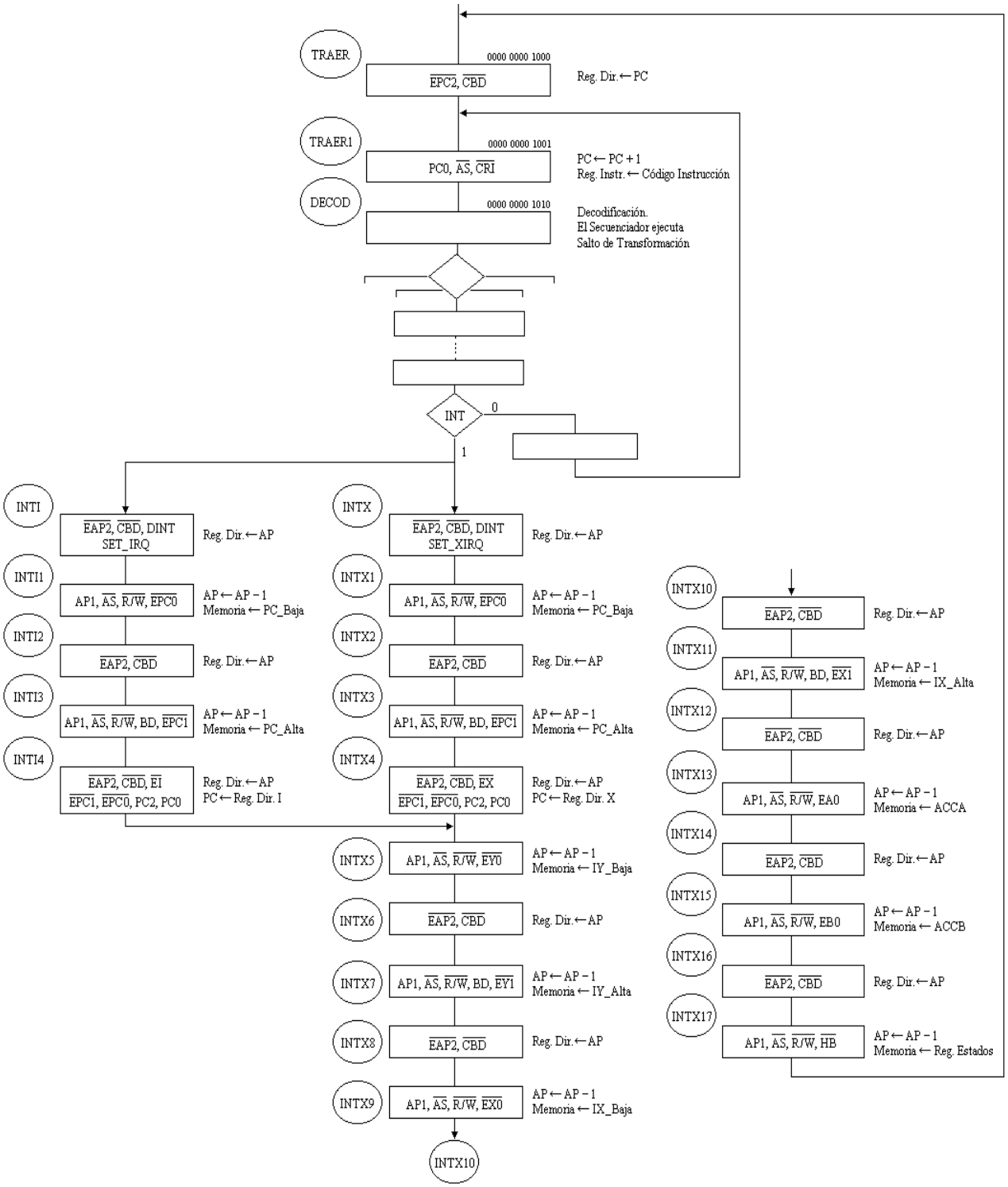


Figura 6.29. Carta ASM para atender interrupciones.

En el penúltimo estado de cada instrucción en ensamblador se revisa la bandera de interrupciones (INT). Esta variable valdrá uno si ha ocurrido una interrupción, es decir, si se cumplen las siguientes dos condiciones: 1) hubo una transición de un nivel lógico alto a un nivel lógico bajo en las líneas IRQ y/o XIRQ; y 2) en el registro de banderas están habilitadas las interrupciones. Recuerde que las banderas X e I habilitan las interrupciones provenientes de las líneas XIRQ e IRQ, respectivamente.

En este mismo estado el secuenciador ejecuta la instrucción de salto condicional con interrupciones. Si el valor de INT es uno entonces se realizará el salto hacia alguno de los estados de atención a la interrupción, cuya dirección será proporcionada por la Unidad de Control de Interrupciones. Si la línea IRQ fue quien generó la interrupción entonces se salta al estado "INTI"; por el contrario, si fue XIRQ quien generó la interrupción entonces se saltará al estado "INTX".

Cuando se detecta una interrupción es necesario guardar el contenido del registro PC para poderlo restaurar después de atender a la interrupción. Este valor de PC es guardado en la pila, que es una zona de memoria que utiliza el microprocesador para almacenar datos temporales. El valor de PC almacenado en la pila corresponde a la dirección de la siguiente instrucción a ejecutar, de manera que una vez atendida la interrupción, se podrá continuar con la ejecución del programa a partir de dicha instrucción.

Una vez guardada la copia de PC, el PC se carga con el valor de la dirección de inicio de un manejador o driver. Los manejadores o drivers son segmentos de código que atienden las peticiones de los dispositivos que generaron la interrupción. Las direcciones para estos drivers se obtienen de los registros de interrupciones DIRI y DIRX, según la línea que generó la interrupción.

Por ejemplo, suponga que la línea XIRQ fue quien generó la interrupción, por lo tanto, primero se guarda el contenido de PC en la pila y enseguida se carga el PC con la dirección de inicio del manejador para la interrupción XIRQ; esta dirección se obtiene del registro de interrupción DIRX. Los pasos anteriores se ejecutan del estado "INTX" al estado "INTX4". Ahora suponga que la línea IRQ fue quien generó la interrupción, por lo tanto, se guarda el contenido de PC en la pila y se carga el PC con la dirección de inicio del manejador para la interrupción IRQ, la cual se obtiene del registro de interrupción DIRI. Estos pasos se ejecutan del estado "INTI" al estado "INTI4".

Es importante mencionar que en los estados "INTI" e "INTX" se activan las señales SET_IRQ y SET_XIRQ, respectivamente. Estas señales colocan en estado de set a los flip-flops de la unidad de control de interrupciones, lo que permite detectar otras interrupciones para su posterior atención. En estos mismos estados también es activada la señal DINT, la cual deshabilita la generación de la señal INT. De esta manera, es posible detectar nuevas peticiones de interrupción, las cuales serán atendidas hasta que la petición en curso sea terminada[7].

[7] Consulte la Figura 5.12 para mayor información sobre la estructura interna de la Unidad de Control de Interrupciones.

Los siguientes pasos en la carta ASM consisten en guardar el estado de los demás registros de la arquitectura, es decir, son guardados en la pila el contenido del registro índice Y, del registro índice X, del acumulador A, del acumulador B y del registro de banderas. Este procedimiento se realiza del estado “INTX5” al estado “INTX17”.

Finalmente, se comienza un nuevo ciclo fetch utilizando la dirección de inicio del driver, es decir, se trae la primera instrucción que atiende al dispositivo que generó la interrupción.

6.3.12 REGRESO DE INTERRUPCIÓN

Instrucción: RTI

Operación:

AP ⇐(AP) + 1,	⇑(CCR)
AP ⇐(AP) + 1,	⇑(ACCB)
AP ⇐(AP) + 1,	⇑(ACCA)
AP ⇐(AP) + 1,	⇑(IX $_{\text{Alta}}$)
AP ⇐(AP) + 1,	⇑(IX $_{\text{Baja}}$)
AP ⇐(AP) + 1,	⇑(IY $_{\text{Alta}}$)
AP ⇐(AP) + 1,	⇑(IY $_{\text{Baja}}$)
AP ⇐(AP) + 1,	⇑(PC $_{\text{Alta}}$)
AP ⇐(AP) + 1,	⇑(PC $_{\text{Baja}}$)

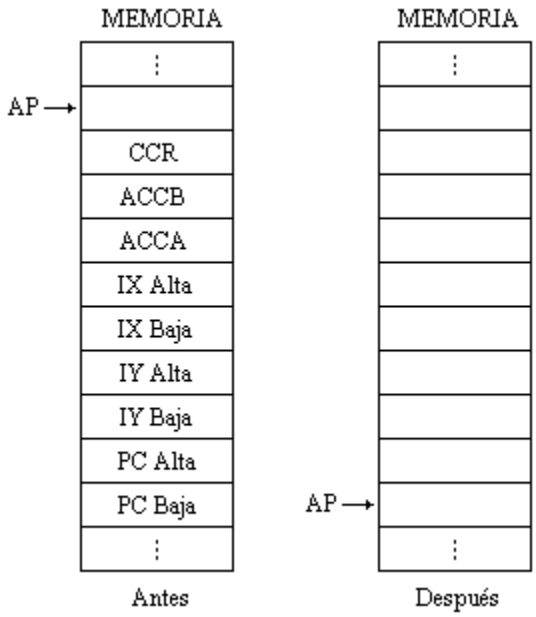
Código: 3B

Descripción: Regreso de Interrupción. Es restaurado el contenido del registro de estados, del acumulador B, del acumulador A, del registro índice X, del registro índice Y, y del contador de programa.

Banderas: Los bits de estado son modificados de acuerdo al valor almacenado en la pila, a excepción de la bandera X, la cual no puede cambiar de cero a uno. Sólo está permitido que cambie de uno a cero, o que mantenga su valor.

S	X	H	I	N	Z	V	C
↕	↓	↕	↕	↕	↕	↕	↕

El contenido de la pila después de ejecutar está instrucción queda de la siguiente manera.



En la figura 6.30 se presenta la carta ASM que ejecuta esta instrucción.

Como es sabido, en los primeros dos estados se obtiene el código de operación de la instrucción, y en el tercero es decodificado dicho código.

En el estado “RTI” se incrementa el apuntador de pila (AP), de manera que apunte a la dirección en memoria en donde se encuentra el contenido del registro de estados. En el siguiente estado, “RTI1”, la dirección guardada en AP se carga en el registro de direcciones. En el estado “RTI2”, se lee de memoria el dato a restaurar y se guarda en el registro de banderas. En este mismo estado también se incrementa el apuntador de pila para continuar restaurando el resto de los registros.

Del estado “RTI3” al estado “RTI14” es restaurado el contenido del acumulador B, del acumulador A, del registro índice X y del registro índice Y; y del estado “RTI15” al estado “RTI18” se restaura el contenido de PC para continuar con la ejecución del programa. Recuerde que antes de atender a la interrupción se guardó el valor de PC con la dirección de la siguiente instrucción a ejecutar.

Cuando se trataron las interrupciones[8] se mencionó que durante la atención a una interrupción es desactivada la generación de la línea INT, de esta manera, sólo se registran las nuevas peticiones de interrupción, las cuales serán atendidas hasta que la interrupción actual ha concluido; es decir, hasta que se ejecute una instrucción RTI. Por lo tanto, en el estado “RTI17” también se activa la señal HINT, para que nuevamente se pueda generar la línea INT y así atender a alguna interrupción que se haya solicitado mientras se atendía la interrupción actual.

[8] Para mayor información consulte la figura 6.29 y el apartado de atención a interrupciones.

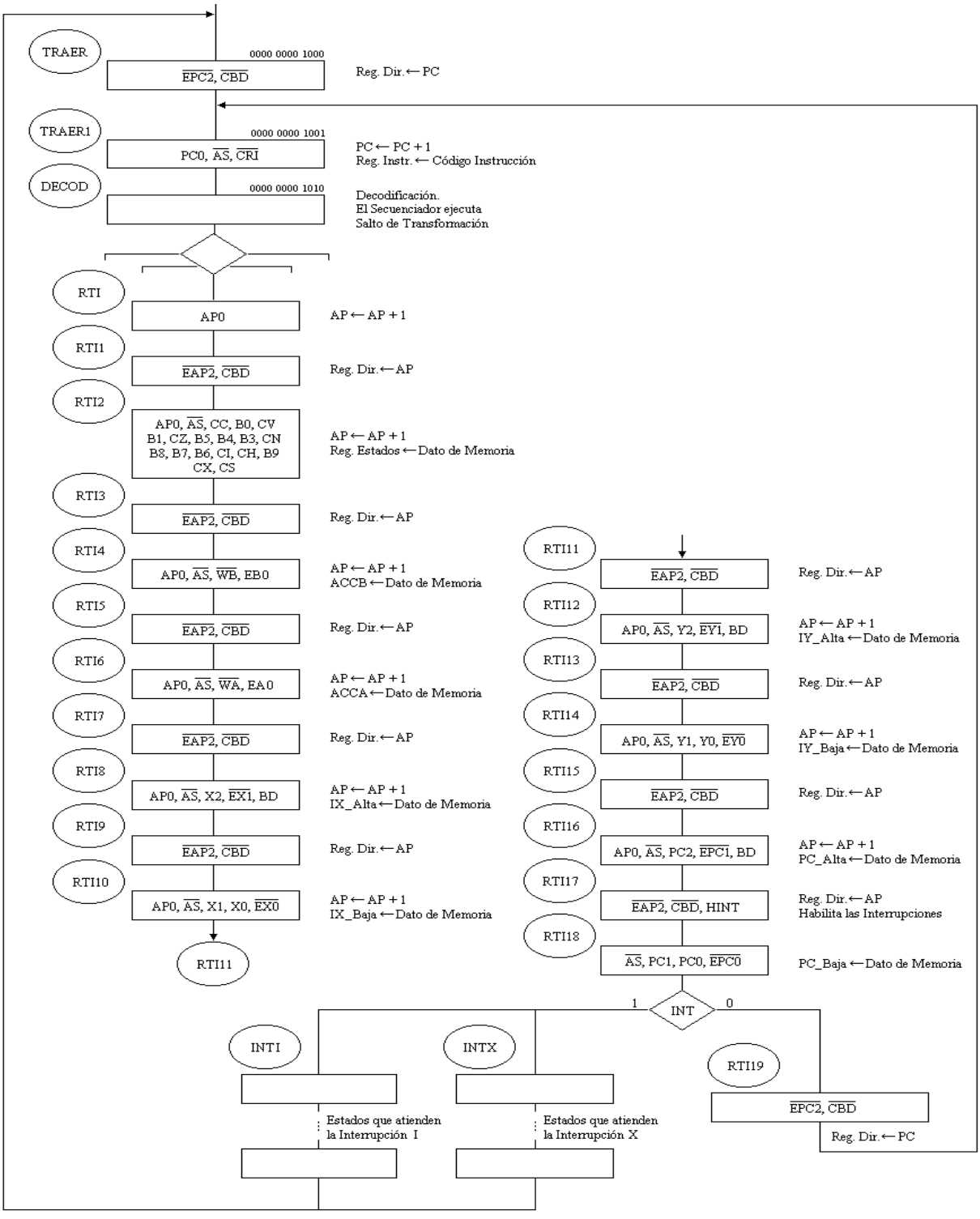


Figura 6.30. Carta ASM para la instrucción RTI (acceso inherente).

PROBLEMAS

1. Construya la carta ASM para la instrucción TXS (acceso inherente).

   Instrucción: TXS
   Operación: SP ⇐IX – 1
   Código: 35
   Descripción: Se carga en el registro apuntador de pila el contenido del registro índice IX menos uno. El contenido de IX no se modifica.
   Banderas: Ninguna bandera es afectada.

2. Construya la carta ASM para la instrucción TSTA (acceso inherente).

   Instrucción: TSTA
   Operación: ACCA – 0
   Código: 4D
   Descripción: Resta al contenido del acumulador A el valor de cero. Esta instrucción sólo actualizada banderas en el registro de estados.
   Banderas: Son actualizadas las banderas N, Z, V=0 y C=0.

3. Construya la carta ASM para la instrucción LDAA (acceso directo).

   Instrucción: LDAA Dirección_8Bits
   Operación: ACCA ⇐(Memoria)
   Código: 96
   Descripción: Carga en el acumulador A, un dato inmediato de 8 bits contenido en memoria.
   Banderas: Son actualizadas las banderas N, Z, V=0.

4. Construya la carta ASM para la instrucción ABA (acceso inherente).

   Instrucción: ABA
   Operación: ACCA ⇐ACCA + ACCB
   Código: 1B
   Descripción: Suma los contenidos de los registros acumuladores A y B. El resultado es guardado en el acumulador A.
   Banderas: Son actualizadas las banderas H, N, Z, V, C.

5. Construya la carta ASM para la instrucción BNE (acceso relativo).

   Instrucción: BNE Desplazamiento
   Operación: Si Z=0, PC ⇐(PC) + 2 + Desplazamiento
   Código: 26

Descripción: Salto condicional. Si Z=0 entonces PC ⇐(PC) + 2 + Desplazamiento, si Z=1 entonces PC ⇐(PC) + 2.
Banderas: Ninguna bandera es afectada.

6. Construya la carta ASM para la instrucción PSHX (acceso inherente).

Instrucción: PSHX
Operación: ⇓IX_{BAJA}, SP ⇐SP – 1
⇓IX_{ALTA}, SP ⇐SP – 1
Código: 3C
Descripción: El contenido del registro índice IX es guardado en la pila. El apuntador de pila (SP) se decrementa cada vez que guardamos un byte en la pila.
Banderas: Ninguna bandera es afectada.

7. Construya la carta ASM para la instrucción ABX (acceso inherente).

Instrucción: ABX
Operación: IX ⇐IX + ACCB
Código: 3A
Descripción: Suma el contenido del acumulador B (un dato de 8 bits sin signo) al contenido del registro índice IX. El resultado es guardado en el registro IX.
Banderas: Ninguna bandera es afectada.

8. Construya la carta ASM para la instrucción COM (acceso extendido).

Instrucción: COM Dirección_16Bits
Operación: Memoria ⇐ Memoria = 0xFF – Memoria
Código: 73
Descripción: Reemplaza el dato contenido en memoria por su complemento a uno.
Banderas: Son actualizadas las banderas N, Z, V=0 y C=1. Si desea calcular el complemento a uno de un número sin afectar la bandera de acarreo, entonces, ejecute la operación OR-exclusiva entre el número deseado y el valor 0xFF.

9. Construya la carta ASM para la instrucción BCLR (acceso directo).

Instrucción: BCLR Dirección_8Bits Máscara
Operación: Memoria ⇐(Memoria) • Máscara
Código: 15
Descripción: Primero se lee un dato de la localidad de memoria Dirección_8Bits. Los bits de ese dato se limpian según el valor de la máscara de 8 bits, y el resultado obtenido es guardado nuevamente en Dirección_8Bits.
Banderas: Son actualizadas las banderas N, Z y V=0.