CAPITULO

6

Conceptos de programación

En el capítulo anterior quedó claro que un programa no es más que un conjunto de instrucciones almacenadas en memoria, que dirige las actividades del microprocesador. Los programas de computación se denominan habitualmente "software". Escribir un programa requiere sólo papel y lápiz; una vez que el programa ha sido escrito (y funciona) puede colocarse en una memoria ROM, según ya se ha detallado. La memoria ROM que contiene el programa suele denominarse "firmware"; una vez grabado el programa en la memoria ROM, es muy difícil modificarlo. Debe pensarse en la ROM como si fuera una tarjeta perforada; una vez perforados los agujeros en la tarjeta, es muy difícil alterar su contenido sin tener que usar una tarjeta nueva.

6.1 Diagramas de flujo

Antes de comenzar a escribir un programa, es práctica común generar un "diagrama de flujo" de las actividades que se desea que el programa ejecute.

DIAGRAMA DE FLUJO: Representación gráfica que ilustra los pasos lógicos, cálculos y decisiones, y la secuencia en que deben ejecutarse los mismos, para llevar a cabo una tarea específica.

El diagrama de flujo de la Fig. 6.1 ilustra los pasos y decisiones tomados por el modelo de computadora descripto en el capítulo 5. Las flechas indican la dirección adoptada luego de la ejecución de cada instrucción o bloque de decisión. La Fig. 6.2 ilustra los símbolos básicos utilizados en los diagramas de flujo.

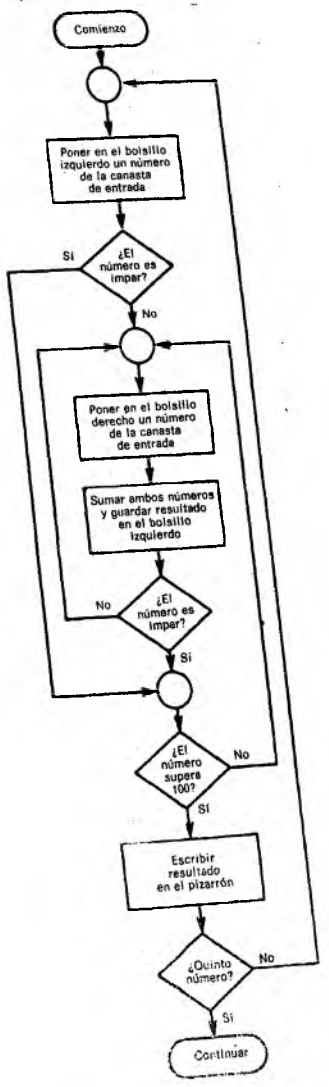


Fig. Diagrama de flujo

Luego de generar un diagrama de flujo, es relativamente simple escribir el programa. Este hecho normalmente debe ser comprobado por el programador en forma personal.

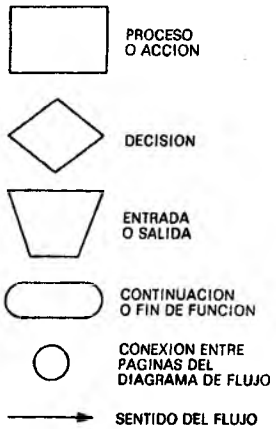


Fig. 6.2 Símbolos del diagrama de flujo

6.2 Códigos nemotécnicos

Según se ha visto, las computadoras digitales solamente operan con números binarios, no entienden frases en inglés o cualquier otro idioma. No obstante, la mayor parte de los fabricantes definen para cada instrucción un código de dos, tres o cuatro letras, que describe elementalmente la función de la misma. Este código "nemotécnico" tiene un equivalente binario para representar la función. En el modelo de computadora desarrollado en el capítulo 5, una de las instrucciones era "sumar el contenido del acumulador A con el contenido del acumulador B y dejar el resultado en el acumulador A". Si se pretende escribir un programa, aunque sea simple, y cada instrucción requiere semejante definición, la tarea llevaría años. Para simplificar, el fabricante asigna un código nemotécnico para describir cada una de las instrucciones. El código nemotécnico en este caso es ABA (sumar A con B). De nuevo, debe recordarse que este código no es más que un elemento de ayuda para el programador. La operación ABA puede representarse por el valor hexadecimal 1B, que en la memoria ROM aparece como 0001 1011. Cuando el MP encuentra esta instrucción, la decodifica interpretando, "sumar los contenidos de los dos acumuladores, colocar el resultado en el acumulador A, y buscar la próxima instrucción en la dirección de memoria siguiente".

CODIGO NEMOTECNICO: Código simple, habitualmente alfabético, representativo de la función de la instrucción codificada.

6.3 Ensambladores

Puede surgir la siguiente pregunta: ¿Cómo se llega del código nemotécnico al lenguaje binario que la computadora puede interpretar? Los programas expresados con códigos nemotécnicos suelen llamarse "programas fuente". El código binario se obtiene a partir de estos programas fuente.

LENGUAJE DE MAQUINA: Otra denominación para el lenguaje binario (unos y ceros), que es el único lenguaje que puede entender una computadora digital.

Una vez que se ha escrito el programa fuente para ejecutar una tarea, su conversión al lenguaje de máquina puede realizarse de dos maneras diferentes. Una manera es guiarse por el manual de programación del MP, buscando el equivalente binario de cada código nemotécnico del programa fuente. Esta tarea suele ser muy tediosa, especialmente por el tiempo que consume y porque algunas traducciones pueden requerir cálculos en los cuales no se admite error. Esta técnica "manual" de traducción o ensamble es muy utilizada por el aficionado, pero no sirve para aplicaciones profesionales.

El segundo método de traducción de código nemotécnico a lenguaje de máquina es utilizar un programa "ensamblador". Un programa ensamblador es un programa diseñado para convertir un programa fuente (escrito con códigos nemotécnicos) a lenguaje de máquina (Fig. 6.3). Cada tipo de microprocesador existente en el mercado debe tener un programa ensamblador especialmente diseñado para el mismo, dado que las arquitecturas circuitales y los códigos nemotécnicos varían de un MP a otro. Pueden, no obstante, obtenerse programas ensambladores para distintos microprocesadores que pueden ejecutarse sobre un mismo sistema. Esto significa que podría ejecutarse una traducción o ensamble de lenguaje fuente del M6800 a lenguaje de máquina del mismo

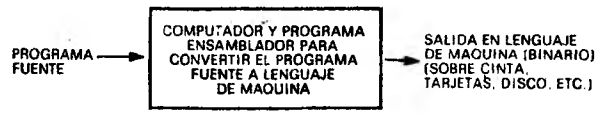


Fig. 6.3 Codigos ASCII

MP sobre un computador IBM 360. Este mismo IBM 360 puede ejecutar un programa traductor escrito por otra compañía para convertir los programas fuente de su MP a lenguaje de máquina. El programa ensamblador del M6800 se analizará en más detalle en el capítulo 8.

6.4 Código ASCII

La sigla ASCII, correspondiente a American Standard Code for Information Interchange, surge tarde o temprano cuando se trabaja

CARACTER	CODIGO ASCII	CARACTER	CODIGO ASCII
@	1000000	CAMBIO DE HOJA	0001100
A	1000001	RETORNO DE CARRO	0001101
B	1000010	BORRAR	1111111
C	1000011	ESPACIO	0100000
D	1000100	!	0100001
E	1000101	"	0100010
F	1000110	#	0100011
G	1000111	$	0100100
H	1001000	%	0100101
I	1001001	&	0100110
J	1001010	'	0100111
K	1001011	(	0101000
L	1001100	)	0101001
M	1001101	*	0101010
N	1001110	+	0101011
O	1001111	,	0101100
P	1010000	–	0101101
Q	1010001	.	0101110
R	1010010	/	0101111
S	1010011	0	0110000
T	1010100	1	0110001
U	1010101	2	0110010
V	1010110	3	0110011
W	1010111	4	0110100
X	1011000	5	0110101
Y	1011001	6	0110110
Z	1011010	7	0110111
[	1011011	8	0111000
\	1011100	9	0111001
]	1011101	:	0111010
↑	1011110	;	0111011
NULO	0000000	<	0111100
TAB HORIZ	0001001	=	0111101
CAMB LINEA	0001010	>	0111110
TAB VERT	0001011	?	0111111

Fig. 6.4 Códigos ASCII (7 bits)

en este tema. Se trata de un código utilizado para el intercambio de información entre dispositivos como impresoras, máquinas de escribir, etc., y el microprocesador. En la Fig. 6.4 se observa que el código ASCII de la letra "A" es "1000001". Esto significa que, si se desea imprimir una letra "A" en la impresora del sistema, debe enviarse a esa impresora, por las líneas de datos, el conjunto de señales "1000001".

Problemas

1. ¿Qué es un diagrama de flujo?
2. Dibujar un diagrama de flujo que describa la preparación de una torta según la siguiente receta:

   Ingredientes

   6 huevos batidos;
   4 tazas de azúcar impalpable;
   1½ tazas de manteca derretida;
   4 tazas de harina;
   ½ taza de jugo de ananá;
   1 cucharadita de té de esencia de vainilla.

   Mezclar los huevos y el azúcar; agregar la manteca y mezclar. Agregar la harina y amasar bien. Agregar el jugo de ananá y amasar bien. Agregar la esencia de vainilla y mezclar. Colocar en horno frío; calentar el horno hasta 200 grados. Hornear durante 1 hora y 15 minutos.
3. Describir la diferencia entre un programa en lenguaje de máquina, un programa en lenguaje fuente y un programa ensamblador.

CAPITULO

7

Modos de direccionamiento

En el capítulo 5 se desarrolló un modelo de computadora muy simplificado por medio del cual se ilustraron los conceptos básicos de los sistemos microcomputadores. Debe recordarse que las instrucciones se ejecutan en secuencia, una a una, a menos que alguna instrucción obligue a una bifurcación o salto a otra dirección de memoria. A medida que se ejecutan las instrucciones, pueden requerir datos. Estos datos están en alguna parte de la memoria, a veces inmediatamente después de la instrucción.

Todas las posiciones de memoria del sistema M6800, objeto de nuestro estudio, tienen ocho dígitos binarios. Por consiguiente, cada instrucción del sistema tiene ocho bits y ocupa una posición de memoria. A pesar de que cada instrucción tiene ocho bits (o un byte) de ancho, habitualmente requiere el uso de una o dos posiciones de memoria subsiguientes.

BIT: Dígito binario (8 dígitos binarios = 8 bits)
BYTE: Grupo de 8 dígitos binarios (8 bits)

Si una instrucción requiere el uso de la posición de memoria siguiente, se la denomina instrucción de dos bits. Por ejemplo, suponga se que luego de una instrucción de cargar el acumulador A viene el número binario 10011101, que debe ser cargado en ese acumulador. La memoria ROM tendrá la distribución indicada en la Fig. 7.1. Luego de que el MP haya ejecutado la instrucción en la dirección hexadecimal 8000, suponiendo que es una instrucción de un byte, deberá ejecutar la instrucción de la dirección 8001.

DIRECCION: Forma de llamar comúnmente a las posiciones de memoria. La posición de memoria 8001 se podrá denominar "dirección 8001". Todas las direcciones de memoria mencionadas en este capítulo aparecen en hexadecimal, salvo que se especifique lo contrario.

Por supuesto, no debe olvidarse que la instrucción de la dirección 8001 está en binario. Cuando el microprocesador decodifica esta instrucción, la interpreta como: "tomar el número almacenado en la dirección siguiente (8002), colocarlo en el acumulador A, y luego ir a la dirección 8003 para leer la próxima instrucción".

DIRECCION DE MEMORIA	CONTENIDO
8000	
8001	Cargar acumulador A con cont. de dirección 8002
8002	10011101
8003	

Fig. 7.1 Dirección de memoria

La pregunta que surge ahora es: ¿Cómo sabe el MP que su próxima instrucción está en la dirección 8003 y no en la 8002? La respuesta es simple: al decodificar la instrucción almacenada en la dirección 8001, parte de la información almacenada en esa dirección le indica al MP que su próxima instrucción está en 8003.

Algunas instrucciones requieren dos direcciones subsiguientes para poder ejecutarse. Por ejemplo, las dos direcciones siguientes pueden contener la dirección de memoria en la que se encuentra almacenado el operando a utilizar. Si el número 11100111 debe cargarse en el acumulador A, estando este número almacenado en la dirección de memoria 9150, la memoria ROM aparece como se ilustra en la Fig. 7.2. Al decodificar la instrucción de la dirección 8001 el MP interpreta: "tomar el número almacenado en la dirección 9150, colocar ese número en el acumulador A, y luego ir a la dirección 8004 a leer la próxima instrucción".

Puede apreciarse que la instrucción de la Fig. 7.2 es una instrucción de tres bytes, a pesar de que la instrucción en sí sólo requiere un único byte en la memoria.

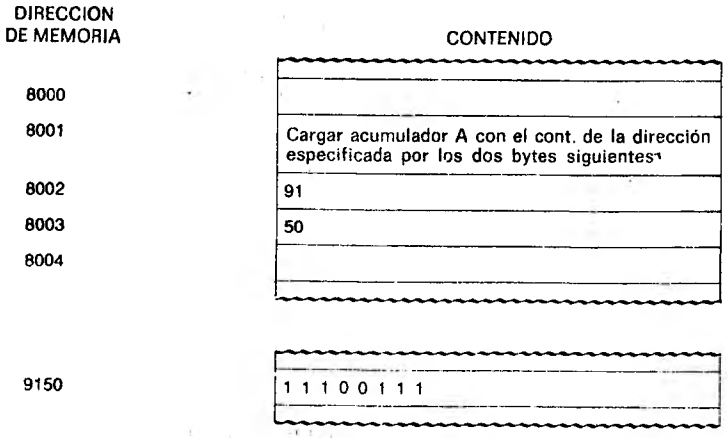


Fig. 7.2 Dirección de memoria (cont.)

Ambos ejemplos llevan al mismo resultado final, esto es, cargar un dato en el acumulador A. Sin embargo, en cada uno de los dos casos fue necesario tomar la información de una zona de memoria diferente. Cada uno de los dos ejemplos opera en un "modo" diferente para llegar al mismo resultado final. En este capítulo se describirán todos los "modos de direccionamiento" del M6800. Muchas instrucciones pueden operar en uno, dos o tres modos diferentes, para llegar al mismo resultado final.

El M6800 tiene siete modos de direccionamiento:

1. Inherente (o implícito)
2. Acumulador
3. Inmediato
4. Directo
5. Extendido
6. Indexado
7. Relativo

Cada uno de estos modos se ilustrarán con ejemplos.

7.1 Registros del microprocesador

Antes de analizar los modos de direccionamiento del M6800, se hará un repaso de los registros internos y acumuladores disponibles en su unidad de procesamiento. Los mismos se analizarán en mucho mayor detalle en el capítulo 9; no obstante, se hace necesario un conocimiento elemental de los mismos para poder interpretar los modos de direccionamiento descriptos en este capítulo.

Acumuladores A y B

> **Acumulador:** también llamado "registro". Es decir, acumulador A y registro A significan lo mismo.

Los acumuladores A y B son registros de ocho bits ubicados en la unidad de procesamiento que se utilizan para manejo de datos, almacenamiento temporario y otras funciones lógicas del MP.

Registro índice

El registro índice tiene dieciséis bits (dos bytes). Se usa principalmente para modificar direcciones de memoria.

Contador de programa

El contador de programa es un registro de dieciséis bits (dos bytes) que contiene la dirección del próximo bit que debe leerse en memoria. Se utiliza para mantener el control del programa.

Puntero de pila

El puntero de pila (stack pointer) es un registro de dieciséis bits que contiene una dirección de memoria, en la cual puede almacenarse, bajo ciertas condiciones, el estado de los registros del MP.

Registro de código de condición

Este registro tiene ocho bits y se utiliza para verificar los resultados de ciertas operaciones.

7.2 Modo de direccionamiento (inherente o implícito)

En el modo de direccionamiento inherente, la instrucción no requiere para su ejecución dirección alguna de memoria. Las instrucciones expresadas en modo inherente siempre tienen un solo byte. Se las suele denominar "instrucciones inherentes". Ejemplo de instrucción implícita es la instrucción "INX". El concepto de esta instrucción

es "sumar 1 al registro índice". Después de la ejecución de esta instrucción, el registro índice de dieciséis bits habrá sido incrementado en una unidad.

7.3 Modo de direccionamiento por acumulador

Las instrucciones expresadas en este modo de direccionamiento son instrucciones de 1 byte y direccionan alguno de los dos acumuladores del MP. Ejemplo de este tipo es la instrucción COM A (complementar el acumulador). Luego de la ejecución de esta instrucción, cada uno de los ocho bits del registro A habrá sido invertido, esto es, todos los unos convertidos a ceros y viceversa.

7.4 Modo de direccionamiento inmediato

En el modo de direccionamiento inmediato los datos se encuentran en la primera o las dos primeras direcciones de memoria que siguen al código de operación. Ejemplo de este tipo sería la instrucción que carga el acumulador B con el número $F0_{16}$ (11110000). La instrucción que ejecuta esta operación se codifica en memoria como $C6_{16}$ (11000110). Nótese, según se aprecia en la Fig. 7.3, que cualquier

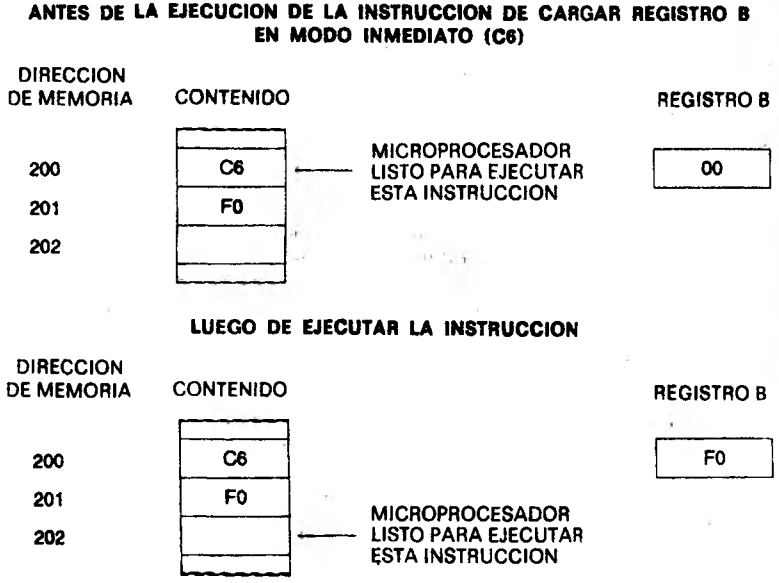


Fig. 7.3 Modo de direccionamiento inmediato

cosa almacenada en el registro acumulador B se perderá, siendo reemplazado por el número $F0_{16}$. Nótese además que, luego de cargar el registro B con el número $F0_{16}$, el MP sabe que su próxima instrucción está almacenada en la dirección 202. Al decodificar el contenido de la dirección 200, el MP pudo deducir, a partir del código de operación C6, que la próxima instrucción está en la dirección 202. Este tipo de instrucción puede tener tres bytes. En el caso de tener que cargar un dato en el registro índice o en el puntero de pila, ese dato deberá ocupar dos bytes, dado que ambos registros son de dieciséis bits. Cuando se carga un operando en los registros de dieciséis bits del MP, este operando ocupa las dos palabras de memoria siguientes al código de operación.

7.5 Modo de direccionamiento directo

En el modo de direccionamiento directo, la dirección de memoria siguiente al código de operación contiene la dirección en la que se encuentra el operando. En el microprocesador M6800 esta instrucción ocupa dos bytes. Por ejemplo: se pretende cargar el acumulador B con el valor $F0_{16}$, almacenado en la dirección de memoria 78, según se ilustra en la Fig. 7.4. El código de operación de esta instrucción es de $D6_{16}$, o sea 11010110. Debe notarse, nuevamente, que la información contenida en el registro B se pierde, y que al igual que en el modo inmediato, una vez codificada la instrucción representada por el número $B6_{16}$, luego de ejecutarse la instrucción almacenada en la dirección 201, se busca la próxima instrucción en la dirección 203. Véase también que la ejecución de la instrucción no afecta la información almacenada en la dirección de memoria 78. Un detalle importante es que en este modo la dirección del operando debe especificarse con un solo byte (ocho bits); esto significa que en el modo directo de direccionamiento el MP puede acceder a direcciones de memoria que se encuentran entre los valores 00_{16} y FF_{16}.

7.6 Modo de direccionamiento extendido

El modo de direccionamiento directo requiere la especificación de una dirección de solamente ocho bits (un byte). Muy a menudo esto no es posible, debido a que se necesitan dos bytes para su especificación; en esos casos se utiliza el modo de direccionamiento extendido. Este modo es muy similar al de direccionamiento directo, con la diferencia de utilizar tres bytes de memoria, en lugar de dos (uno para el código de operación y otros dos para especificar la dirección de memoria del operando). Como ejemplo, si se desea cargar en el acumulador B el operando $F0_{16}$ almacenado en la dirección

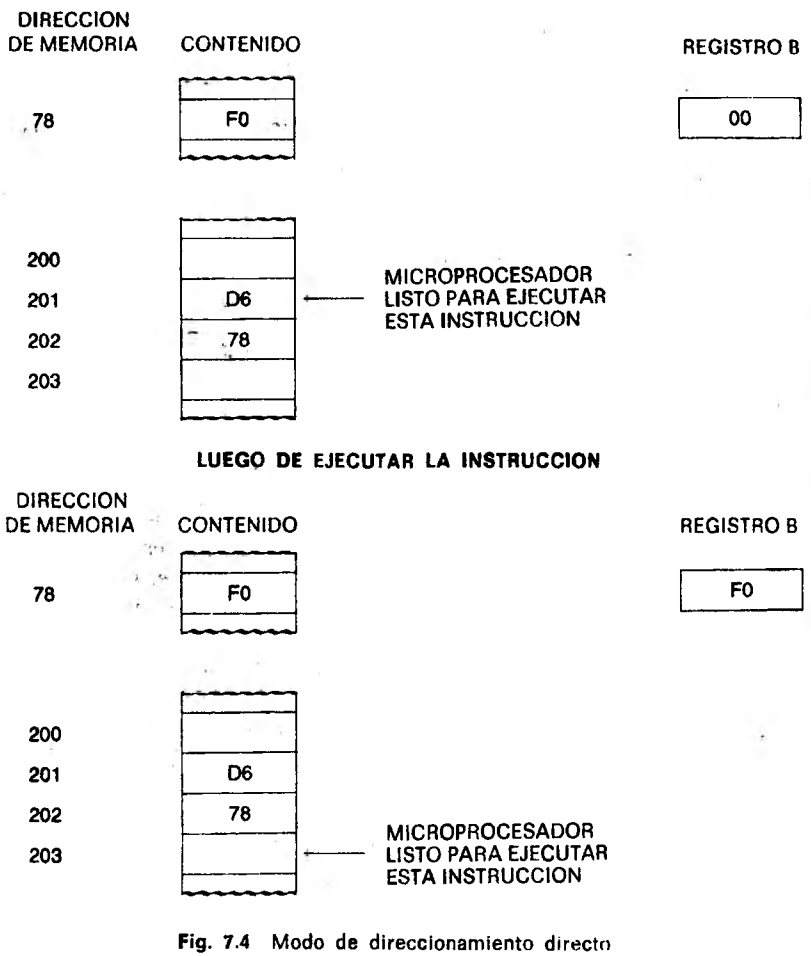


Fig. 7.4 Modo de direccionamiento directo

9051_{16}, el código de operación correspondiente será $F6_{16}$. De la Fig. 7.5 surge que en este modo de direccionamiento la dirección del operando se encuentra almacenada en los dos bytes siguientes al código de operación. El primero de estos dos bytes contiene la mitad más significativa de la dirección, y el segundo la mitad menos significativa.

Nuevamente, luego de la decodificación de la instrucción F6, el microprocesador sabe que su próxima instrucción (después de cargar

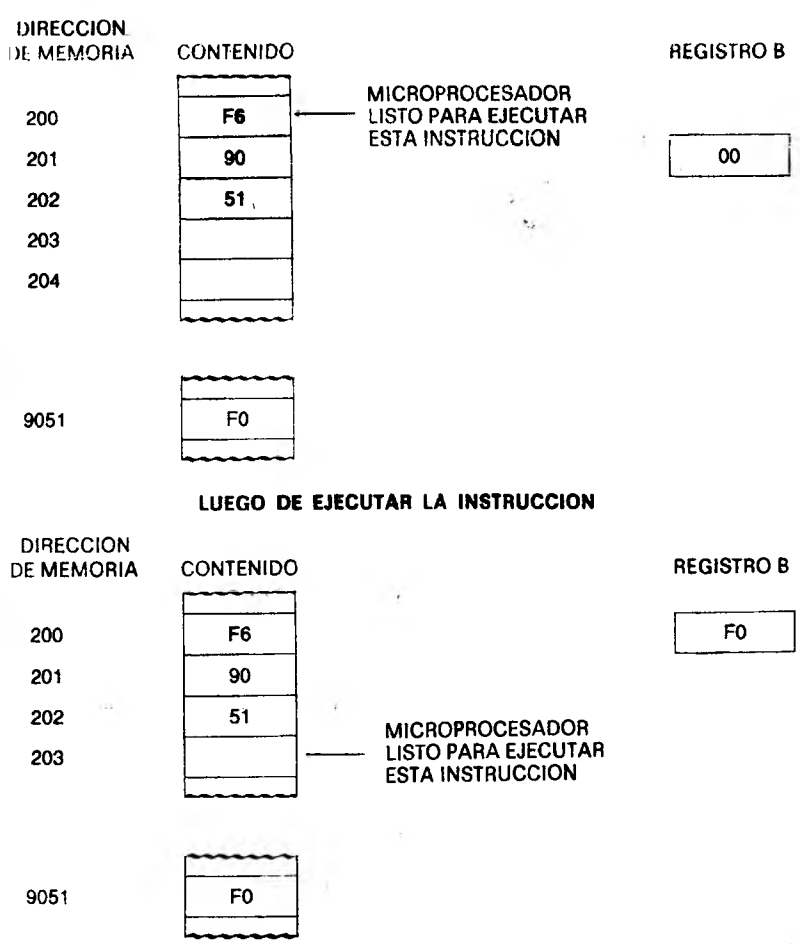


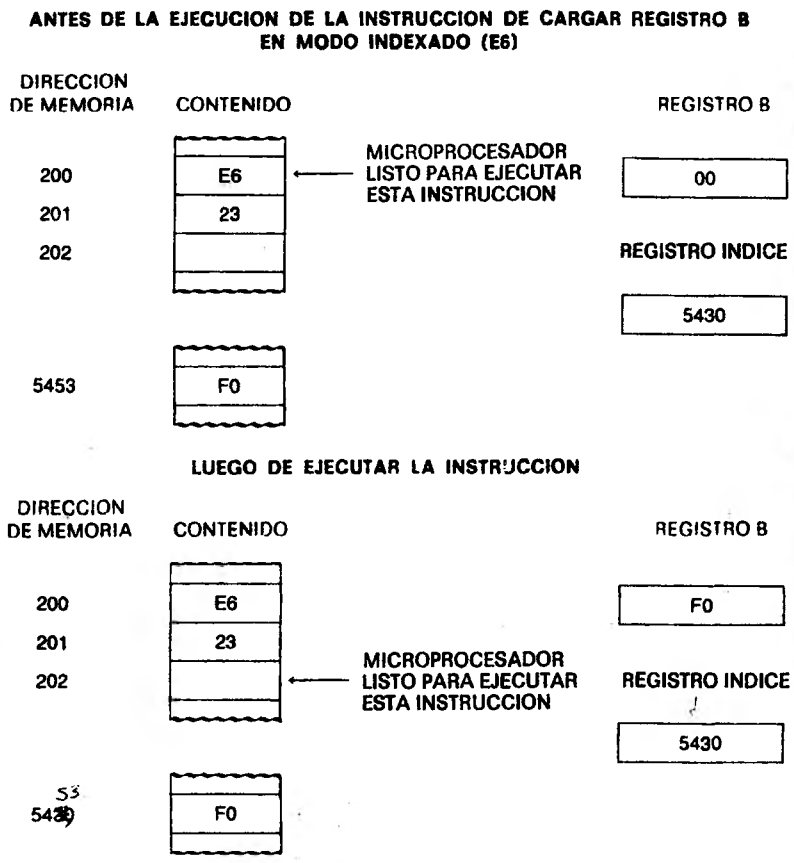
Fig. 7.5 Modo de direccionamiento extendido

el operando de la dirección 9051 en el acumulador B) debe obtenerse de la dirección 203. Nótese además que la dirección 9051 no se altera.

7.7 Modo de direccionamiento indexado

El modo de direccionamiento indexado utiliza una instrucción de dos bytes. La dirección siguiente a la del código de operación con-

tiene un número (comúnmente conocido como "desplazamiento") que, sumado al contenido del registro índice, permite encontrar la dirección del operando. Esta dirección, al igual que en el modo de direccionamiento extendido, contiene el operando buscado. Como ejemplo (Fig. 7.6), supóngase que el contenido del registro índice es 5430_{16}. Si se pretende cargar el acumulador B con el contenido de la dirección 5453_{16}, la instrucción a utilizar para permitir la carga del acumulador deberá tener como código de operación el valor $E6_{16}$.



Recordar que la dirección efectiva se obtiene sumando el contenido del registro índice al contenido del byte siguiente al de la instrucción. En este caso, resulta 5430 + 23 = 5453.

Fig. 7.6 Modo indexado de direccionamiento

en tanto que la dirección de memoria siguiente contendrá el valor 23_{16}, dado que la suma de 23 + 5430 forma 5453, que es la dirección del operando. Debe notarse que, dado que el registro índice es de dieciséis bits, la dirección que se genera es también de dieciséis bits.

El microprocesador decodifica la instrucción igual que en los casos anteriores, y "sabe" que su próxima instrucción está almacenada en la dirección 202. Este modo de generar dirección parece igual al modo de direccionamiento extendido, lo cual no es cierto. Dado que el registro índice puede incrementarse o decrementarse en uno, puede accederse a información en direcciones consecutivas de memoria con sólo incrementar o decrementar ese registro. Esta aplicación podrá verse en programas de ejemplo en capítulos posteriores. Además, el modo indexado requiere sólo un byte aparte del código de operación, mientras que el modo extendido requiere dos.

7.8 Modo de direccionamiento relativo

En el M6800 se utiliza este modo de direccionamiento cuando la siguiente instrucción a ser ejecutada por el microprocesador no se encuentra a continuación de la instrucción en ejecución. Es el caso de las instrucciones denominadas "de bifurcación" (branch). Estas instrucciones hacen que, de acuerdo al estado de alguna condición almacenada en el registro de código de condición, el MP tenga que ir a buscar su próxima instrucción en alguna otra dirección de memoria, rompiendo la secuencia del programa. Estas condiciones se analizarán una por una para cada instrucción, en un capítulo posterior. Por el momento, solamente se analizará el mecanismo de cálculo de la dirección correspondiente a la próxima instrucción del programa a ejecutar.

Las instrucciones de salto ocupan dos bytes. La posición de memoria siguiente al código de operación indica al microprocesador cuál es la dirección de memoria de la próxima instrucción, si se cumplen ciertas y determinadas condiciones. Dado que en este modo de direccionamiento pueden hacerse saltos hacia adelante y hacia atrás, se analizarán ambos casos por separado.

Salto hacia adelante

Como se ha mencionado, la dirección siguiente al código de operación contiene la información que indica al MP cómo saltar a la dirección de la próxima instrucción. Cuando el MP interpreta una instrucción como una bifurcación, la decodifica como instrucción de dos bytes. Si no se produjese la bifurcación a una dirección diferente, la próxima instrucción a ejecutar estaría ubicada en la dirección actual más dos. Por lo tanto, para calcular la dirección a la que se va a saltar, debe

considerarse como punto de partida la dirección de la instrucción de bifurcación más dos. En consecuencia, el MP toma el contenido de la posición de memoria que sigue al código de operación de la instrucción de bifurcación y lo suma a dicha dirección más dos. El resultado de esta suma indica la dirección de la próxima instrucción, nuevo punto de partida para la ejecución de las instrucciones siguientes.

A modo de ejemplo, la Fig. 7.7 representa una instrucción de bifurcación incondicional ubicada en la dirección 0200. Se desea saltar a la dirección 0225. La dirección 0200 contiene el código de la instrucción de bifurcación, que en este caso es 20_{16}. En la dirección de

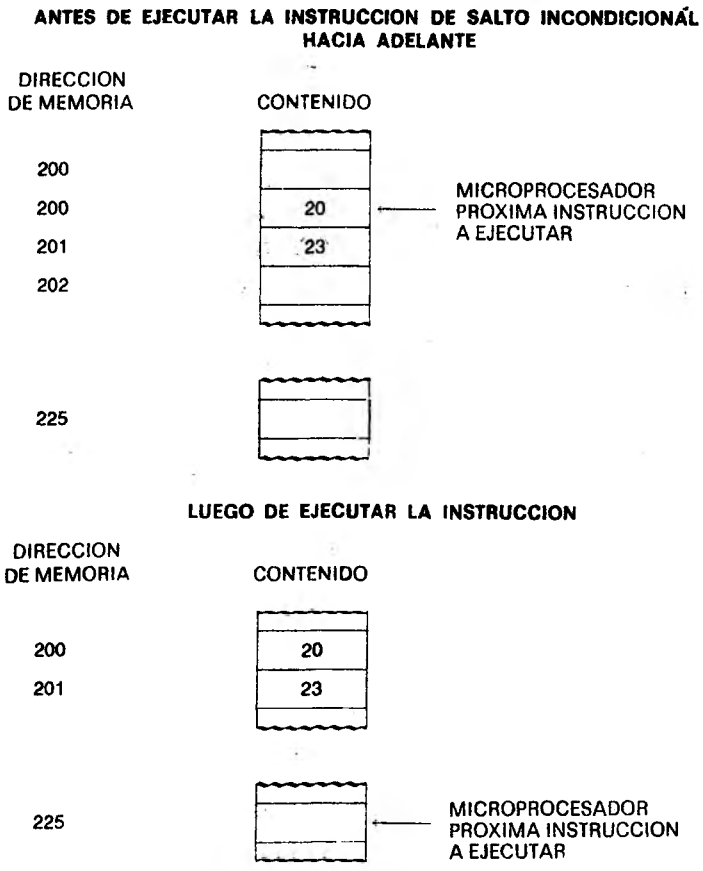


Fig. 7.7 Modo de direccionamiento relativo (salto hacia adelante)

memoria 0201 se almacena el número 23_{16}; el resultado de la suma previamente mencionada (200 + 2 + 23) es dirección 0225 de la próxima instrucción.

Cuando la bifurcación es hacia adelante, el bit más significativo del dato debe ser "0". Por consiguiente, restan solamente siete bits para definir las distintas direcciones de memoria a las cuales el microprocesador puede saltar. Esto significa que el máximo número de direcciones que el MP puede saltar hacia adelante (siempre contando desde la instrucción siguiente a la de salto) es de 127_{10} direcciones, o sea $7F_{16} = 1111111_2$.

Salto hacia atrás

El MP puede saltar hacia atrás desde la dirección de la instrucción de bifurcación, tan fácilmente como hacia adelante. Nuevamente, la dirección de referencia para el salto corresponde a la dirección del código de operación más dos. El dato que acompaña a la instrucción de salto, igual que en el caso anterior, indica al microprocesador la dirección de la próxima instrucción tras la bifurcación. No obstante, este dato debe venir expresado en forma de complemento a 2. Si, por ejemplo, se desea bifurcar desde la dirección 0200 hacia la dirección 0195, deben realizarse los siguientes cálculos para obtener el desplazamiento:

$$\begin{array}{ll} \text{Dirección actual} + 2 & 0202_{16} \\ \text{Dirección final} & -\ 0195_{16} \\ & \overline{6D_{16}} \end{array}$$

(Si la resta en hexadecimal resulta difícil, puede verificarse el resultado restando en binario.)

$$\begin{array}{rl} 6D_{16} = & 01101101 \\ \text{complemento a 1} = & 10010010 \\ \text{complemento a 2} = & 10010011 \end{array}$$

↑

Este "1" indica que se bifurca hacia atrás.

Para que en la próxima instrucción se lea la dirección 0195, el número que sigue a la instrucción de bifurcación debe ser 10010011, es decir, 93_{16} (Fig. 7.8).

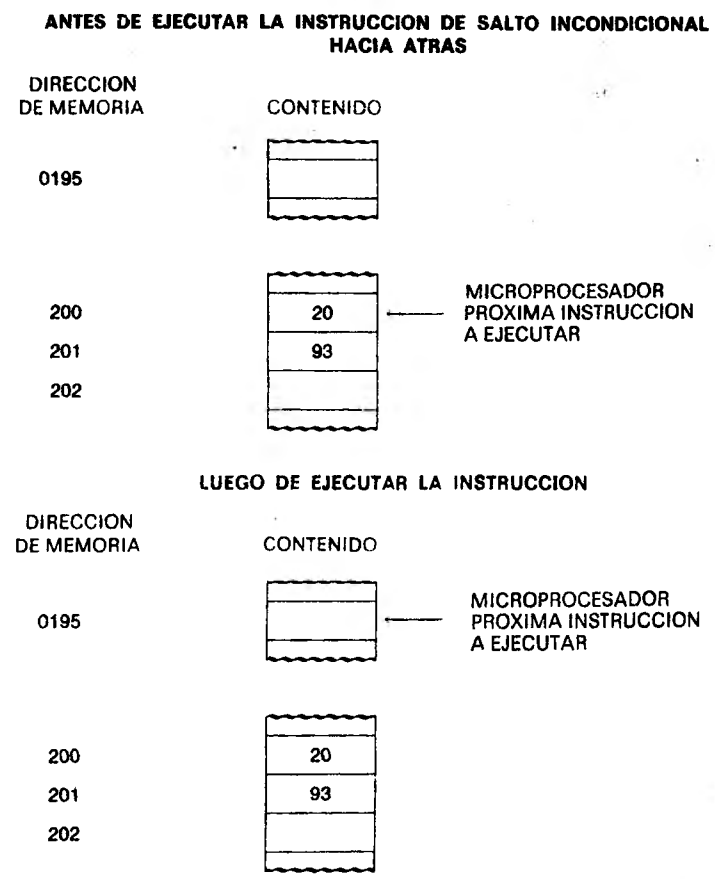


Fig. 7.8 Modo de direccionamiento relativo (salto hacia atras)

7.9 Resumen

Se utilizó como ejemplo la instrucción de carga del acumulador B en cuatro modos de direccionamiento diferentes: inmediato, directo, extendido e indexado. Debe notarse que el código hexadecimal de cada modo de direccionamiento es diferente. Luego de todo este análisis surge claramente lo siguiente: si al escribir un programa, el operando está ubicado en uno o dos bytes siguientes al código de operación, este código de operación debe expresarse en su modo *inmediato*. Si el operando se almacena en una dirección que ocupa un solo byte, debe utilizarse el código de operación para el modo *directo*.* Si la información buscada está en una dirección que no puede expresarse con el modo directo por especificarse con dos bytes, se utilizará el código de operación correspondiente al modo *extendido*. Si se puede especificar la dirección del operando por medio de la suma del contenido del registro índice con algún parámetro expresable en un byte, este parámetro puede ir en la dirección siguiente al código de operación, y éste corresponderá al modo *indexado*. Las instrucciones de *bifurcación* tienen su sistema propio. Según indica la Fig. 7.9, los códigos de operación para una misma instrucción (en este caso cargar el acumulador B) varían de acuerdo al modo de direccionamiento utilizado.

MODO	CODIGO DE INSTRUCCION
CARGAR REGISTRO B MODO INMEDIATO	C6
CARGAR REGISTRO B MODO DIRECTO	D6
CARGAR REGISTRO B MODO EXTENDIDO	F6
CARGAR REGISTRO B MODO INDEXADO	E6

Fig. 7.9 Modos de direccionamiento para la carga del registro B

Problemas

1. ¿Qué modo de direccionamiento se utiliza para las instrucciones de bifurcación?
2. ¿Cuál es la diferencia entre el modo de direccionamiento inmediato y el modo de direccionamiento directo?
3. Si se desea cargar el acumulador A en modo indexado, el registro índice contiene el número 4201_{16} y la dirección de memoria siguiente al código de operación contiene al número 10_{16}, ¿cuál es la dirección de memoria en la que se encuentra el operando?
4. ¿Cuántas direcciones pueden saltarse hacia adelante y hacia atrás?
5. Si la dirección 4050 contiene una instrucción de bifurcación incondicional a la dirección 4080, ¿cuál es el contenido de la dirección 4051?
6. Si en el problema anterior se deseara saltar a la dirección 4000, ¿cuál sería el contenido de 4051?

* En realidad, las direcciones ocupan siempre dos bytes. En el modo directo puede accederse a direcciones en el rango 0000_{16} a $00FF_{16}$; como el primer byte es siempre 00_{16}, se lo da por sobreentendido y se almacena solamente el segundo byte (00_{16} a FF_{16}). *N. del T.*

7. ¿Cuál es la diferencia entre el modo de direccionamiento directo y el modo de direccionamiento extendido?
8. Calcular las direcciones finales de las instrucciones de bifurcación siguientes:

Dirección hexadecimal	*Contenido hexadecimal*	*Dirección de destino*
4014	20	
4015	F8	_____
5535	20	
5536	14	_____
800F	20	
8010	E6	_____
99AB	20	
99AC	F2	_____

CAPITULO

8

Software del M6800

Como se ha visto en el capítulo 7, una instrucción puede estar expresada en uno, dos, tres o incluso cuatro modos de direccionamiento distintos. Si bien puede haber varios modos diferentes para cada instrucción, el resultado final de la instrucción es siempre el mismo. Cada modo de direccionamiento sólo representa una forma distinta de indicarle a la unidad de proceso dónde encontrar la información.

El juego de instrucciones del M6800 incluye setenta y dos instrucciones diferentes; sin embargo, dado que cada instrucción puede tener más de un modo de direccionamiento, existe ciento noventa y siete códigos de operación válidos. Por ejemplo, la instrucción ADDA tiene cuatro códigos de operación diferentes, a saber: El código 8B para el modo inmediato, 9B para el modo directo, BB para el modo extendido y AB para el modo indexado. En el modo inmediato el operando se encuentra en la dirección de memoria siguiente al código de operación; en los otros tres modos de direccionamiento el operando puede estar en cualquier otra zona de memoria.

Este capítulo presenta una revisión de todo el juego de instrucciones del M6800, con un resumen de los modos de direccionamiento de cada instrucción, y el código de operación hexadecimal correspondiente en cada caso. Se presentarán numerosos ejemplos indicando los contenidos de los registros afectados, antes y después de la ejecución de la instrucción. Se da también un listado ejemplificativo para todas las instrucciones en cada modo. La indicación de modo inmediato se realiza mediante el signo #. Si aparece un signo # la instrucción está representada en el modo inmediato, y el número que acompaña al signo # se encuentra en la dirección de memoria siguiente al código de operación. El signo $ indica un número hexadecimal. Si no aparece el signo #, el número que acompaña al signo $ es una direc-

ción de memoria expresada en hexadecimal. El modo de direccionamiento indexado se representa con un número hexadecimal seguido de una coma y de una X mayúscula ($10,X). El valor que acompaña al signo $ se suma al contenido del registro índice para obtener la dirección efectiva de memoria. Por ejemplo:

1. *LDA A #$25* indica el operando hexadecimal 25_{16} (*modo inmediato*).
2. *LDA A $25* indica que el número hexadecimal 25_{16} es una dirección de memoria (*modo directo*).
3. *LDA A $2525* indica la dirección hexadecimal 2525_{16} (*modo extendido*).
4. *LDA A $25,X* indica que la dirección se obtendrá sumando 25_{16} al contenido del registro índice (*modo indexado*).

El microprocesador M6800 tiene un registro de códigos de condición, cuyos bits se ponen en "1" o en "0" de acuerdo con el resultado de determinadas instrucciones. Los ocho bits del registro de códigos de condición son los siguientes:

Bit 0: C (Carry-Borrow) (arrastre en operaciones de suma o resta).

Bit 1: V (Indicador de rebalse en operaciones en complemento a 2).

Bit 2: Z (Indicador de 0).

Bit 3: N (Indicador de negativo).

Bit 4: I (Máscara de interrupciones).

Bit 5: H (Half-Carry) (arrastre desde el bit 3 al 4 de un resultado).

Bits 6 y *7:* Siempre en "1".

En la descripción de cada instrucción se incluye un listado de los bits del registro de códigos de condición afectados por la ejecución de la misma. Si se indica la letra correspondiente a un bit determinado, la Fig. 8.1 aclara lo que ocurre con ese bit. Si alguna de las explicaciones de la Fig. 8.1 no fuese aplicable a una instrucción en particular, la explicación correcta se detallará al final de la instrucción.

H: En uno si hubo arrastre desde el bit 3; en cero en los demás casos.

I: En cero.

N: En uno si el bit más significativo del resultado está en uno; en cero en los demás casos.

Z: En uno si todos los bits del resultado están en cero; en cero en los demás casos.

V: En uno si como resultado de la operación se produjo rebalse en complemento a 2; en cero en los demás casos.

C: En uno si hubo arrastre desde el bit más significativo del resultado; en cero en los demás casos.

Fig. 8.1 Registro de códigos de condición

8.1 Juego de instrucciones del M6800

Esta sección contiene el juego completo de instrucciones del microprocesador M6800. Todas las direcciones y ejemplos utilizados, incluyendo contenidos de memoria y registros, han sido seleccionados al azar con fines puramente ilustrativos. Los paréntesis significan "contenido de". Por ejemplo, la expresión (acumulador A) significa "el contenido del acumulador A". En muchos ejemplos se utiliza una flecha para señalar la próxima instrucción a ser ejecutada por el MP. Las tres letras entre comillas al comienzo de cada instrucción corresponden al código nemotécnico de la misma.

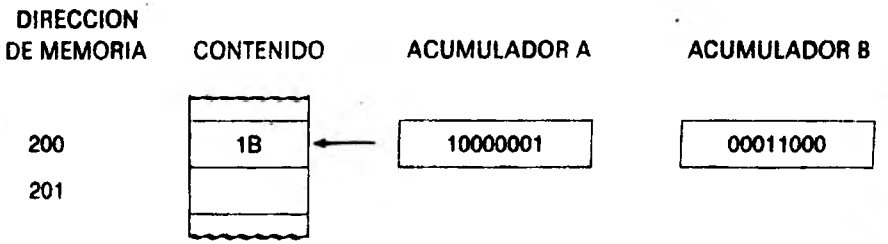
"ABA": Sumar contenido de acumulador B al acumulador A

Se suman el contenido del acumulador A y el contenido del acumulador B, y el resultado se almacena en el acumulador A. El acumulabor B no se modifica.

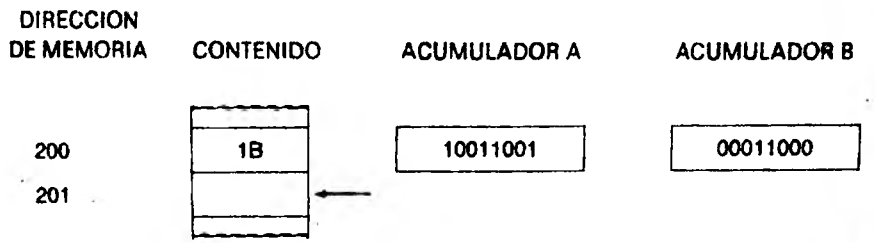
Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	1B	ABA

Los bits del registro de códigos de condición afectados son H, N, Z, V, C (ver Fig. 8.1).

ANTES DE LA EJECUCION DE LA INSTRUCCION ABA



DESPUES DE LA EJECUCION DE LA INSTRUCCION ABA



"ADC": Sumar con arrastre

El contenido del bit C del registro de códigos de condición se suma al contenido del acumulador que corresponda y una dirección de memoria, colocándose al resultado en el mismo acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	89	ADC A #25	Nota 1
A DIRECTO	3	99	ADC A $25	Nota 2
A EXTENDIDO	4	B9	ADC A $7168	Nota 3 (ver ejemplo)
A INDEXADO	5	A9	ADC A $25,X	Nota 4
B INMEDIATO	2	C9	ADC B #$CE	Nota 5
B DIRECTO	3	D9	ADC B $AD	Nota 6
B EXTENDIDO	4	F9	ADC B $CCCC	Nota 7
B INDEXADO	5	E9	ADC B $D2,X	Nota 8

Los bits del registro de códigos de condición afectados son H, N, Z, V, C (ver Fig. 8.1).

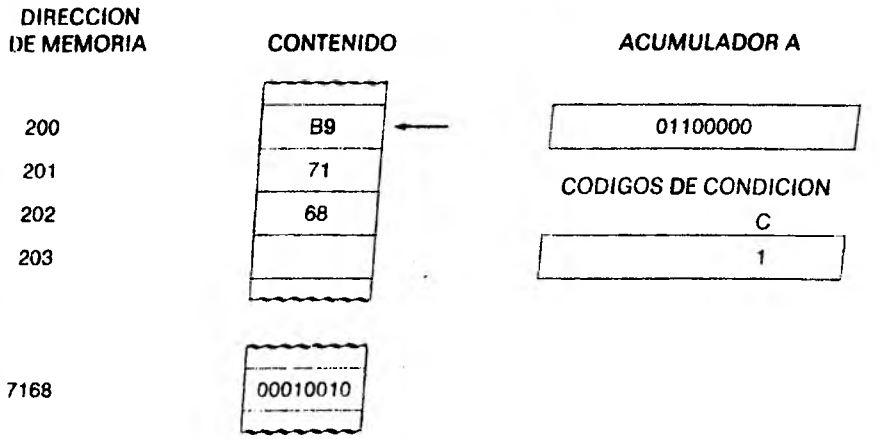
NOTAS:

1. (C) + (acumulador A) + 25_{16}
2. (C) + (acumulador A) + (dirección 25_{16})
3. (C) + (acumulador A) + (dirección 7168_{16})
4. (C) + (acumulador A) + [dirección especificada por (registro índice) + 25_{16}]
5. (C) + (acumulador B) + 25_{16}
6. (C) + (acumulador B) + (dirección AD_{16})
7. (C) + (acumulador B) + (dirección $CCCC_{16}$)
8. (C) + (acumulador B) + [dirección especificada por (registro índice) + $D2_{16}$]

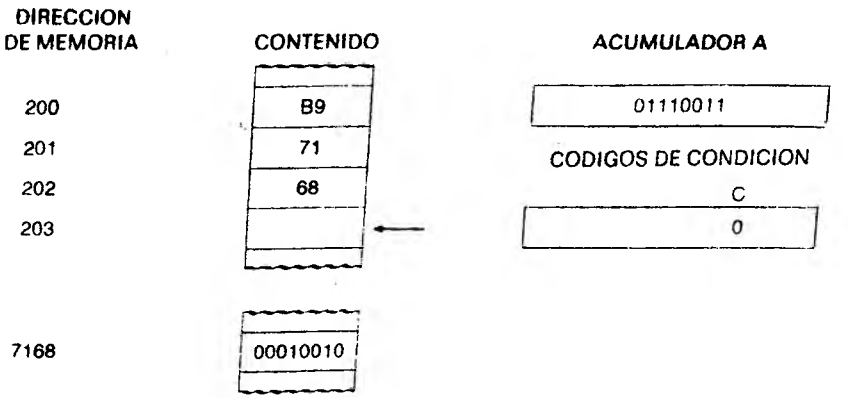
Se coloca el resultado en el acumulador A

Se coloca el resultado en el acumulador B

ANTES DE LA EJECUCION DE LA INSTRUCCION ADC (EJEMPLO DE MODO EXTENDIDO - ADC A $7168)



LUEGO DE LA EJECUCION DE LA INSTRUCCION



"ADD": Sumar sin arrastre

El contenido del acumulador A o del acumulador B se suma al contenido de una dirección de memoria, dejando el resultado en el mismo acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	8B	ADD A #$DA	Nota 9
A DIRECTO	3	9B	ADD A $DA	Nota 10
A EXTENDIDO	4	BB	ADD A $DA53	Nota 11
A INDEXADO	5	AB	ADD A $DA,X	Nota 12
B INMEDIATO	2	CB	ADD B #$21	Nota 13
B DIRECTO	3	DB	ADD B $4D	Nota 14
B EXTENDIDO	4	FB	ADD B $ADFF	Nota 15
B INDEXADO	5	EB	ADD B $55,X	Nota 16

El registro de códigos de condición se afecta en los bits H, N, Z, V, C (ver Fig. 8.1).

NOTAS:

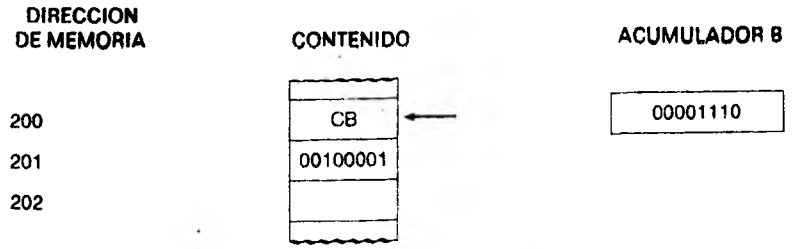
9. (acumulador A) + DA_{16}
10. (acumulador A) + (dirección DA_{16})
11. (acumulador A) + (dirección $DA53_{16}$)
12. (acumulador A) + [dirección especificada por (registro índice) + DA_{16}]

} Se coloca el resultado en el acumulador A

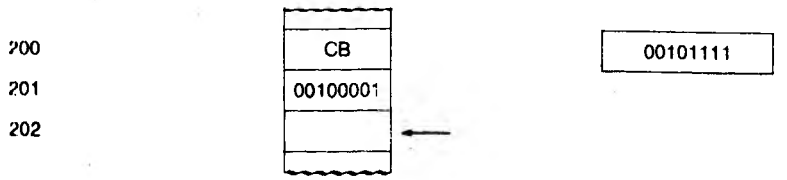
13. (acumulador B) + 21_{16}
14. (acumulador B) + (dirección $4D_{16}$)
15. (acumulador B) + (dirección $ADFF_{16}$)
16. (acumulador B) + [dirección especificada por (registro índice) + 55_{16}]

} Se coloca el resultado en el acumulador B

ANTES DE LA EJECUCION DE LA INSTRUCCION ADD (EJEMPLO UTILIZANDO ADD B #21)



LUEGO DE LA EJECUCION DE LA INSTRUCCION



"AND": Operación lógica "Y"

Realiza la operación lógica "Y" entre cada uno de los bits del acumulador A o del acumulador B y los bits correspondientes del contenido de una posición de memoria, dejando el resultado en el mismo acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	84	AND A #$C2	Nota 17
A DIRECTO	3	94	AND A $6F	Nota 18
A EXTENDIDO	4	B4	AND A $3DCA	Nota 19
A INDEXADO	5	A4	AND A $F1,X	Nota 20
B INMEDIATO	2	C4	AND B #$10	Nota 21
B DIRECTO	3	D4	AND B $10	Nota 22
B EXTENDIDO	4	F4	AND B $1000	Nota 23
B INDEXADO	5	E4	AND B $10,X	Nota 24

Los bits del registro de códigos de condición afectados por esta operación son N y Z (ver Fig. 8.1); el bit V queda en "0".

NOTAS:

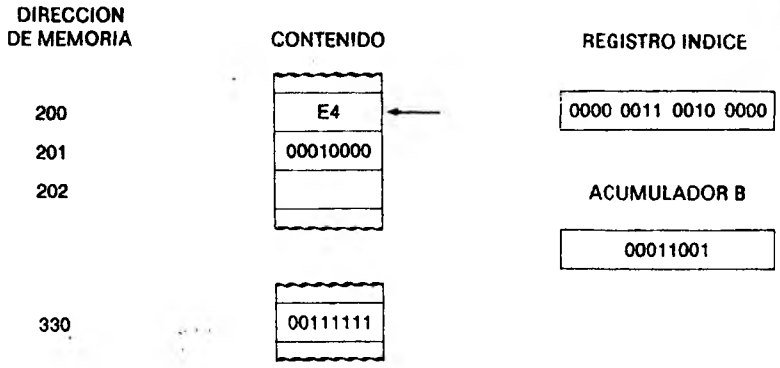
17. (acumulador A) "Y" $C2_{16}$
18. (acumulador A) "Y" (dirección $6F_{16}$)
19. (acumulador A) "Y" (dirección $3DCA_{16}$)
20. (acumulador A) "Y" [dirección definida por (registro índice) + $F1_{16}$]

} Resultado en el acumulador A

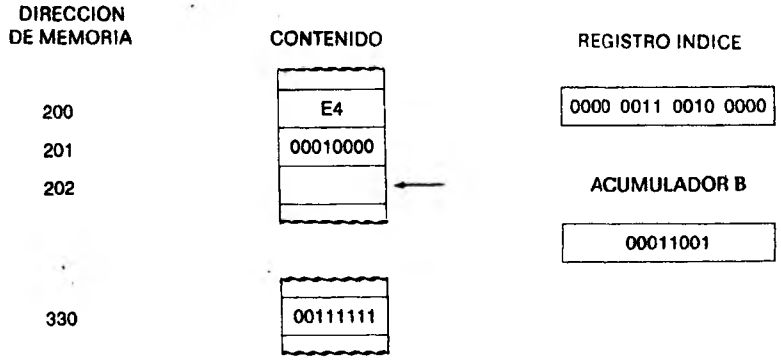
21. (acumulador B) "Y" 10_{16}
22. (acumulador B) "Y" (dirección 10_{16})
23. (acumulador B) "Y" (dirección 1000_{16})
24. (acumulador B) "Y" [dirección definida por (registro índice) + 10_{16}]

} Resultado en el acumulador B

ANTES DE LA EJECUCION DE LA INSTRUCCION AND (EJEMPLO UTILIZANDO AND B $10,X)



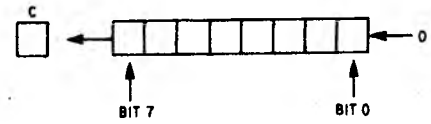
LUEGO DE LA EJECUCION DE LA INSTRUCCION



Puede apreciarse que el contenido del registro índice, sumado al número hexadecimal especificado en el programa (0320 + 10 = 0330) forman la dirección del operando. En la dirección hexadecimal 0330 está el operando (00111111), que al ser sometido a la operación lógica "Y" con el contenido del acumulador B (00011001) dará por resultado 00011001.

"ASL": Desplazamiento aritmético hacia la izquierda (Arithmetic Shift Left)

Se desplazan todos los bits del acumulador A, del acumulador B, o de una dirección de memoria, un lugar hacia la izquierda, colocándose el bit 7 (el más significativo) en el bit C del registro de códigos de condición. En el bit 0 (el menos significativo) del número desplazado se coloca "0".



Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	48	ASL A	Nota 25
ACUMULADOR B	2	58	ASL B	Nota 26
DIRECCION EXTENDIDA	6	78	ASL $67AD	Nota 27 (ver ejemplo)
DIRECCION INDEXADA	7	68	ASL $25,X	Nota 28

Los bits afectados en el registro de códigos de condición son:

N, Z Según Fig. 8.1.

V "1" si luego de completar la operación de desplazamiento N ó C están en "1" (pero no los dos simultáneamente); "0" en caso contrario.

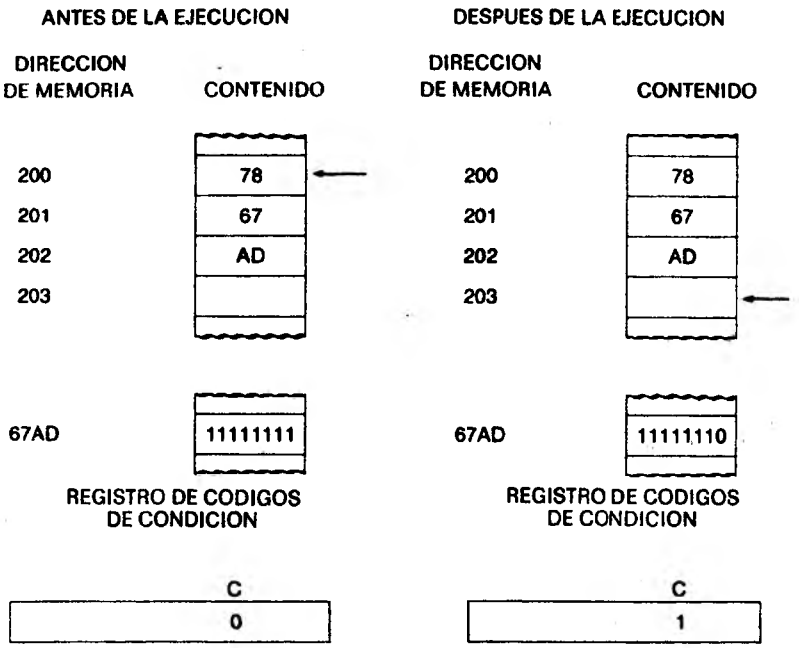
C "1" si el bit más significativo de la dirección a desplazar estaba en "1" antes del desplazamiento; "0" en caso contrario.

NOTAS:

25. (acumulador A) desplazado un lugar a la izquierda
26. (acumulador B) desplazado un lugar a la izquierda
27. (Dirección hex 67AD) desplazado un lugar a la izquierda
28. (Dirección especificada por registro índice + 25) desplazado un lugar a la izquierda

(Bit 7) cargado en el bit C del registro de códigos de condición. Bit 0 cargado con un "0".

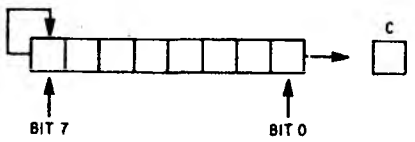
EJECUCION DE LA INSTRUCCION ASL
(EJEMPLO UTILIZANDO ASL $67AD)



El bit 7 de la dirección hexadecimal 67AD se desplaza hacia el bit C, y el bit 0 de la dirección 67AD se carga con un "0".

"ASR": Desplazamiento aritmético a la derecha (Arithmetic Shift Right)

Se desplaza una dirección de memoria, el acumulador A o el acumulador B, un lugar a derecha. El bit 0 se carga en el bit C del registro de códigos de condición. El contenido del bit 7 no se modifica.



Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	47	ASR A	Nota 29 (ver ejemplo)
ACUMULADOR B	2	57	ASR B	Nota 30
DIRECCION EXTENDIDA	6	77	ASR $ABF1	Nota 31
DIRECCION INDEXADA	7	67	ASR $14,X	Nota 32

Los bits afectados en el registro de códigos de condición son:

N, Z Ver Fig. 8.1.

V "1" si luego de la operación de desplazamiento N ó C están en "1" (pero no los dos simultáneamente); "0" en caso contrario.

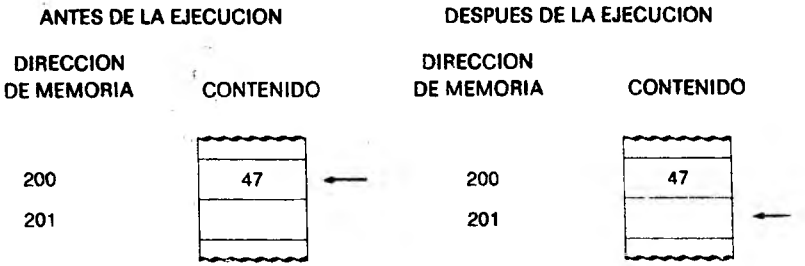
C "1" si el bit menos significativo del número desplazado era "1" antes del desplazamiento; "0" en caso contrario.

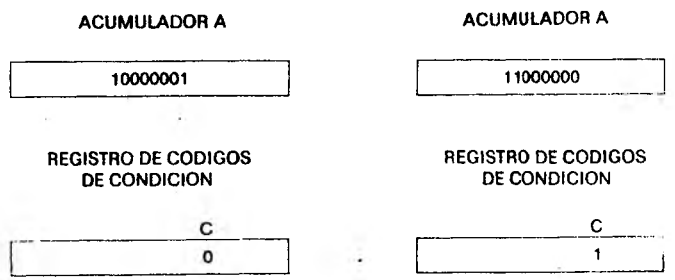
NOTAS:

29. (acumulador A) desplazado un lugar a la derecha
30. (acumulador B) desplazado un lugar a la derecha
31. (Dirección hex ABF1) desplazado un lugar a la derecha
32. (Dirección especificada por registro índice + + hex 14) desplazado un lugar a la derecha

} El bit 0 se almacena en el bit C del registro de códigos de condición; el bit 7 se mantiene inalterado.

EJECUCION DE LA INSTRUCCION ASR
(EJEMPLO UTILIZANDO ASR A)





El bit 0 del acumulador A se coloca en el bit C. El contenido del bit 7 se mantiene inalterado, modificándose el contenido del bit 6 para que coincida con el bit 7.

Instrucciones de bifurcación (Branch)

Todas las instrucciones de bifurcación, excepto la bifurcación incondicional ("Branch Always") y la bifurcación a subrutina ("Branch to Subroutine") dependen del estado de distintos bits del registro de códigos de condición. Estos bits ven alterado su valor de acuerdo con la última instrucción ejecutada por el MP (en realidad, la última instrucción ejecutada que afecte el contenido del registro de códigos de condición) antes de encontrar la instrucción de bifurcación. Si se cumplen las condiciones para la bifurcación, la próxima instrucción a ejecutarse queda especificada por el contenido de la próxima dirección, más la dirección actual, más 2. Si no se cumplen las condiciones de bifurcación, la siguiente instrucción está a continuación de la instrucción de salto (dirección de la instrucción de salto más 2).

Las instrucciones de bifurcación no afectan el contenido del registro de códigos de condición. Por ejemplo, la instrucción BCC (bifurcar si el contenido del bit de arrastre es "0") indica al MP que verifique el contenido del bit C del registro de códigos de condición. Si este bit está en "0", la dirección de la próxima instrucción se obtiene sumando a la dirección de la instrucción de bifurcación el contenido de la dirección siguiente, y sumando 2 al total. Si el registro de códigos de condición contiene "1" en su bit C, no se ejecuta la bifurcación y la próxima instrucción está a continuación de la instrucción de salto (dirección actual más 2).

La Tabla 8.1 ilustra las distintas instrucciones de bifurcación. Nótese que todas ellas son instrucciones de dos bytes (ver sección 8.4 para ejemplos de bifurcaciones condicionales).

Tabla 8.1 Instrucciones de bifurcación

CODIGO NEMOTECNICO	EXPLICACION	TIEMPO DE EJECUCION (CICLOS)	CODIGO DE INSTRUCCION (HEX)	CONDICION DE SALTO	EJEMPLO DE LISTADO FUENTE
BCC	Salto si arrastre 0	4	24	C = 0	BCC
BCS	Salto si arrastre 1	4	25	C = 1	BCS
BEQ	Salto si cero	4	27	Z = 1	BEQ
BGE	Salto si mayor o igual a cero	4	2C	(N = 1 y V = 1) o (N = 0 y V = 0)	BGE
BGT	Salto si mayor que cero	4	2E	(Z = 0 y [(N = 1 y V = 1) o (N = 0 y V = 0)]	BGT
BHI	Salto si superior	4	22	C = 0 o Z = 0	BHI
BLE	Salto si menor o igual a cero	4	2F	(Z = 1) o [(N = 1 y V = 0) o (N = 0 y V = 1)]	BLE
BLS	Salto si inferior o igual	4	23	(C = 1) o (Z = 1)	BLS
BLT	Salto si menor que cero	4	2D	(N = 1 y V = 0) o (N = 0 y V = 1)	BLT
BMI	Salto si negativo	4	2B	N = 1	BMI
BNE	Salto si no cero	4	26	Z = 0	BNE
BPL	Salto si positivo	4	2A	N = 0	BPL
BRA	Salto incondicional	4	20	Nula	BRA $77
BVC	Salto si desborde 0	4	28	V = 0	BVC $4C
BVS	Salto si desborde 1	4	29	V = 1	BVS $DF
BSR	Ver explicación detallada				

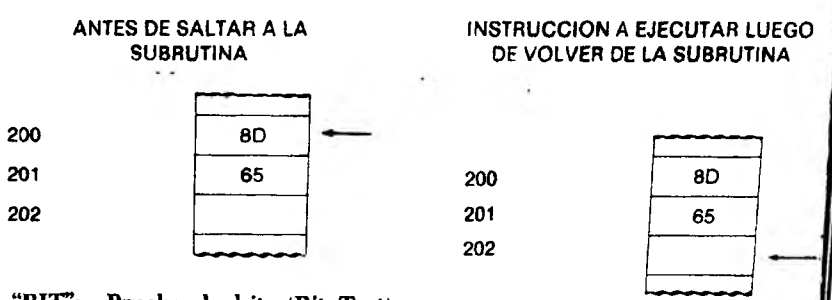
NOTA: Todas las ramificaciones (branches) deben realizarse usando rótulos (labels), o sea BCC TEMP. El ensamblador calculará el desplazamiento (offset, segundo byte de la instrucción) referido a la dirección real de la instrucción de ramificación más 2 (ver ejemplo en sección 8.2 y capítulo 11). Un método alternativo es usar un asterisco (*). En este caso, el ensamblador calculará el "offset" (segundo byte de la instrucción) referido a la dirección real de la instrucción de ramificación (o sea, BCC * + $20).

"BSR": Bifurcación a subrutina (Branch to Subroutine)

Esta instrucción hace que el MP bifurque hacia una subrutina (otro programa) ubicada en alguna otra dirección de memoria. La dirección de la subrutina, en la cual debe encontrarse la próxima instrucción, se determina en la misma forma que en todas las instrucciones de bifurcación. Esta instrucción permite que el programador utilice un mismo subprograma (subrutina) varias veces durante la ejecución de un programa más grande, en lugar de repetir esta sección de programa en memoria cada vez que se la requiera. Al final de la subrutina deberá haber una instrucción RTS que haga retornar al MP a su dirección original, más 2.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
8	8D	BSR $65

BSR $65 hace que el MP bifurque a una dirección de comienzo de subrutina calculada sumando 2 a la dirección de la instrucción BSR, más el valor hexadecimal 65. En el ejemplo que sigue, la subrutina se encuentra en la dirección hexadecimal 267. Luego de detectar una instrucción RTS, el MP regresa a su dirección original más 2.



"BIT": Prueba de bits (Bit Test)

Se realiza la operación lógica "Y" entre cada bit del acumulador A ó B con el bit correspondiente de una dirección de memoria, con el objeto de colocar en "0" o en "1" los bits N y Z (el bit V siempre se pone en cero; los otros bits de condición no se modifican). No se afectan los acumuladores A ó B ni el contenido de la memoria. Esta instrucción se utiliza comúnmente antes de una instrucción de bifurcación, para determinar las condiciones que permitan la ocurrencia de dicha bifurcación.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	85	BIT A #$77	Nota 33
A DIRECTO	3	95	BIT A $77	Nota 34
A EXTENDIDO	4	B5	BIT A $7777	Nota 35
A INDEXADO	5	A5	BIT A $77,X	Nota 36
B INMEDIATO	2	C5	BIT B #$40	Nota 37
B DIRECTO	3	D5	BIT B $FD	Nota 38
B EXTENDIDO	4	F5	BIT B $85F6	Nota 39
B INDEXADO	5	E5	BIT B $20,X	Nota 40

Los bits del registro de códigos de condición se afectan en la forma siguiente:

N "1" si el bit más significativo del resultado de la operación lógica "Y" fuese "1"; "0" en caso contrario.

Z "1" si todos los bits del resultado de la operación lógica "Y" fuesen "0"; "0" en caso contrario.

V "0".

NOTAS:

33. (acumulador A) "Y" hexadecimal 77
34. (acumulador A) "Y" (dirección de memoria hexadecimal 77)
35. (acumulador A) "Y" (dirección hexadecimal 7777)
36. (acumulador A) "Y" (dirección especificada por registro índice + + hexadecimal 77)
37. (acumulador B) "Y" hexadecimal 40
38. (acumulador B) "Y" (dirección hexadecimal FD)
39. (acumulador B) "Y" (dirección hexadecimal 85F6)
40. (acumulador B) "Y" (dirección especificada por registro índice + + hexadecimal 20)

"CBA": Comparar acumuladores

El contenido del acumulador B se resta del contenido del acumulador A, con el objeto de colocar en "0" o en "1" los bits N, Z, V y C del registro de códigos de condición. No se modifica el contenido de ninguno de los dos acumuladores.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	11	CBA

Los bits del registro de códigos de condición afectados son:

N, Z, V (ver Fig. 8.1).

C "1" si la resta requiriese "pedir prestado" un bit para la posición más significativa del resultado; "0" en caso contrario.

"CLC": Borrar el bit de arrastre (Clear Carry)

Esta instrucción coloca el bit C del registro de códigos de condición en "0".

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OC	CLC

El único bit del registro de códigos de condición afectado es el bit C, que adopta el valor "0".

"CLI": Borrar máscara de interrupción (Clear Interrrupt Mask)

Coloca en 0 el bit I del registro de códigos de condición.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OE	CLI

El único bit afectado en el registro de códigos de condición es el bit I, que adopta el valor "0".

"CLR": Borrar (Clear)

Reemplaza por ceros (borra) el contenido del acumulador A, acumulador B o la dirección de memoria especificada.

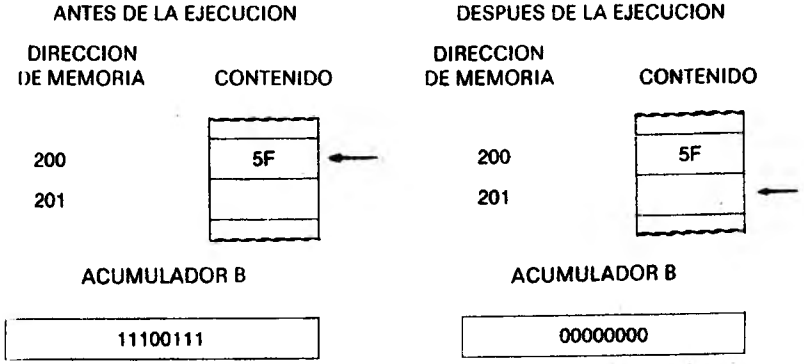
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	4F	CLR A	Nota 41
ACUMULADOR B	2	5F	CLR B	Nota 42 (ver ejemplo)
DIRECCION EXTENDIDA	6	7F	CLR $76D8	Nota 43
DIRECCION INDEXADA	7	6F	CLR $52,X	Nota 44

Los bits del registro de códigos de condición afectados son: N, V y C en "0"; Z en "1".

NOTAS:

41. (acumulador A) borrado
42. (acumulador B) borrado
43. (Dirección hexadecimal 76D8) borrado
44. (Dirección especificada por el registro índice + 52 hexadecimal) borrado

EJECUCION DE LA INSTRUCCION CLR (EJEMPLO UTILIZANDO CLR B)



"CLV": Borrar bit V

Esta instrucción coloca en "0" el bit V del registro de códigos de condición.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OA	CLV

Los bits afectados en el registro de códigos de condición son: V en "0".

"CMP": Comparar

Esta instrucción resta el contenido de una dirección de memoria de alguno de los dos acumuladores, con el objeto de fijar los bits N,

Z, V y C en el registro de códigos de condición. No se modifican los contenidos de la dirección de memoria utilizada ni del acumulador. Esta instrucción suele utilizarse antes de una instrucción de bifurcación condicional, para verificar si debe llevarse a cabo la bifurcación.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	81	CMP A #$52	Nota 45
A DIRECTO	3	91	CMP A $F3	Nota 46
A EXTENDIDO	4	B1	CMP A $AABC	Nota 47
A INDEXADO	5	A1	CMP B $12,X	Nota 48
B INMEDIATO	2	C1	CMP B #$43	Nota 49
B DIRECTO	3	D1	CMP B $75	Nota 50
B EXTENDIDO	4	F1	CMP B $A465	Nota 51
B INDEXADO	5	E1	CMP B $35,X	Nota 52

En el registro de códigos de condición se ven afectados los bits N, Z, V y C (ver Fig. 8.1).

NOTAS:

45 (acumulador A) — hexadecimal 52
46 (acumulador A) — (dirección de memoria hexadecimal F3)
47 (acumulador A) — (dirección de memoria hexadecimal AABC)
48 (acumulador A) — (dirección especificada por registro índice + + hexadecimal 12)
49. (acumulador B) — hexadecimal 43
50 (acumulador B) — (dirección de memoria hexadecimal 75)
51 (acumulador B) — (dirección de memoria hexadecimal A465)
52 (acumulador B) — (dirección especificada por registro índice + + hexadecimal 35)

"COM": Complementar

Esta instrucción toma cada bit del acumulador A, o del acumulador B, o de una dirección de memoria, y lo reemplaza con su complemento a 1. Todos los "1" se reemplazan por "0" y viceversa.

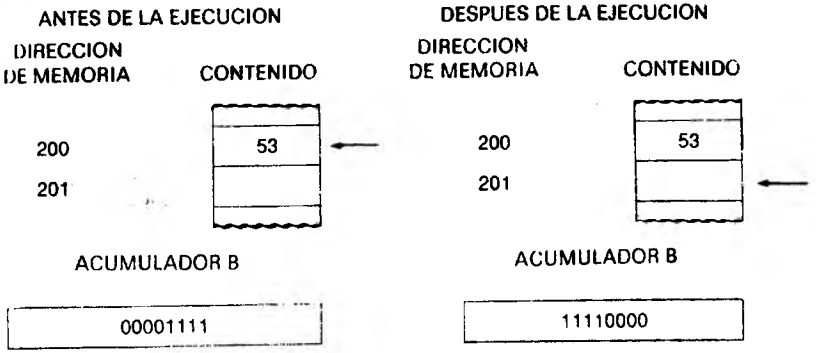
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	43	COM A	Nota 53
ACUMULADOR B	2	53	COM B	Nota 54 (ver ejemplo)
DIRECCION EXTENDIDA	6	73	COM $723F	Nota 55
DIRECCION INDEXADA	7	63	COM $35,X	Nota 56

En el registro de códigos de condición se afectan los bits N y Z según la Fig 8.1, el bit V se borra (se coloca en "0") y el bit C se coloca en "1".

NOTAS:

53. Reemplazar el contenido del acumulador A con su complemento a 1
54. Reemplazar el contenido del acumulador B con su complemento a 1
55. Reemplazar el contenido de la dirección 723F con su complemento a 1
56. Reemplazar el contenido de la dirección hexadecimal especificada por el registro índice + hexadecimal por su complemento a 1

EJEMPLO DE INSTRUCCION COM (COM B)



"CPX": Comparar registro índice

Esta instrucción compara el contenido del registro índice con dos posiciones de memoria consecutivas, dado que el registro índice contiene dieciséis bits y cada dirección de memoria ocho bits. No se modifican los contenidos del registro índice ni de las dos posiciones de memoria involucradas. El contenido de la primera posición de memoria se resta del byte más significativo (bits 8 a 15) del registro índice, y el de la segunda (consecutiva a la anterior) se resta del byte menos significativo (bits 0 a 7) del registro índice.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION INMEDIATA	3	8C	CPX #$543F	Nota 57
DIRECCION DIRECTA	4	9C	CPX $AE	Nota 58
DIRECCION EXTENDIDA	5	BC	CPX $AC52	Nota 59 (ver ejemplo)
DIRECCION INDEXADA	6	AC	CPX $71,X	Nota 60

Los bits afectados en el registro de códigos de condición son:

N "1" si está en "1" el bit más significativo de la resta realizada; "0" en caso contrario.

Z "1" si todos los bits de los resultados de ambas restas fueran "0"; "0" en caso contrario.

V "1" si la resta del byte más significativo del registro índice produce desborde (overflow); "0" en caso contrario.

NOTAS:

57. (Bits 8 a 15 del registro índice) — 54 hexadecimal
(Bits 0 a 7 del registro índice) — 3F hexadecimal

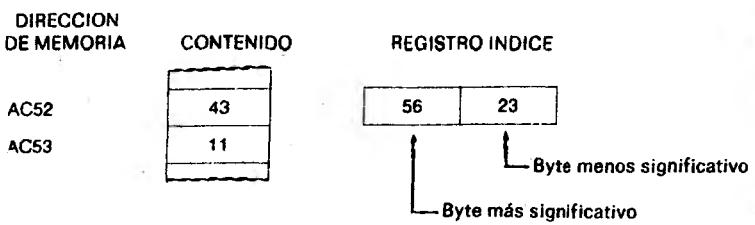
58. (Bits 8 a 15 del registro índice) — (dirección hexadecimal AE)
(Bits 0 a 7 del registro índice) — (dirección hexadecimal AF)

59. (Bits 8 a 15 del registro índice) — (dirección hexadecimal AC52)
(Bits 0 al 7 del registro índice) — (dirección hexadecimal AC53)

60. (Bits 8 a 15 del registro índice) — (dirección especificada por el registro índice + + hexadecimal 71)
(Bits 0 a 7 del registro índice) — (dirección especificada por el registro índice + + hexadecimal 71 + 1)

Los bits N, Z, V del registro de códigos de condición se modifican de acuerdo al resultado obtenido.

EJEMPLO DE INSTRUCCION CPX (CPX $AC52)



La ejecución de la instrucción CPX $AC52 implica la ejecución de las siguientes operaciones hexadecimales:

56 (byte más significativo del registro índice) — 43 (contenido de la dirección AC52) = 13

23 (byte menos significativo del registro índice) — 11 (contenido de la dirección AC53) = 12

Como resultado de las operaciones anteriores, la única acción es la puesta en "0" o en "1" de los bits N, Z y V del registro de códigos de condición. En este ejemplo N = 0, Z = 0 y V = 0.

"DAA": Ajuste decimal del acumulador A (Decimal Adjust A Accumulator)

Cuando se realizan sumas BCD utilizando las instrucciones ABA, ADD y ADC, pueden obtenerse resultados inválidos, ya que las operaciones mencionadas entregan sus resultados (en el acumulador A) en binario.

Por ejemplo, cuando se suman los números decimales 4_{10} y 99_{10}, el resultado es 103_{10}. El número 99_{10} expresado en BCD es 1001 1001, mientras que el número 4_{10} se representa por 0100; por consiguiente:

$$\begin{aligned} 99_{10} \text{ en BCD} &= 1001\ 1001 \\ 4_{10} \text{ en BCD} &= \underline{0000\ 0100} \\ &\quad 1001\ 1101 \end{aligned}$$

Como se ve, el resultado es inválido dado que 1101 no existe en BCD. Sabemos que el resultado debe tener doce bits (103 = 0001 0000 0011). Si al resultado obtenido se le sumase el número decimal 66_{10} se obtendría lo siguiente:

$$\begin{aligned} 99_{10} \text{ en BCD} &= 1001\ 1001 \\ 4_{10} \text{ en BCD} &= \underline{0000\ 0100} \\ &\quad 1001\ 1101 \\ 66_{10} \text{ en BCD} &= \underline{0110\ 0110} \\ &\quad 1\ 0000\ 0011 \\ &\quad 1\ \ 0\ \ \ 3 \end{aligned}$$

Véase cómo la suma de 66_{10} al resultado binario lo corrige para obtener el resultado correcto BCD.

Si se ejecuta la instrucción DAA inmediatamente a continuación de una suma BCD realizada con las instrucciones ABA, ADD o ADC, la ejecución de la instrucción DAA analizará todas las posibles situa-

ciones que puedan dar lugar a resultados inválidos, realizando en cada caso el ajuste correspondiente para obtener el resultado BCD correcto.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	19	DAA

El registro de códigos de condición se ve afectado en sus bits N y Z según la Fig. 8.1; el bit C adopta el valor que correspondería si la instrucción de ajuste decimal y la suma binaria previamente realizada pudiesen reemplazarse por una instrucción hipotética de suma en BCD.

"DEC": Decrementar

El contenido del acumulador A, del acumulador B, o de la dirección de memoria que se indique, se decrementa en una unidad.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	4A	DEC A	Nota 61
ACUMULADOR B	2	5A	DEC B	Nota 62
DIRECCION EXTENDIDA	6	7A	DEC $4956	Nota 63
DIRECCION INDEXADA	7	6A	DEC $19,X	Nota 64

El registro de códigos de condición ve afectados sus bits N y Z según la Fig. 8.1. El bit V se pone en "1" si de la operación resulta un desborde, caso que ocurre únicamente si el acumulador o la dirección de memoria a decrementar contenían el número hexadecimal 80 antes de la operación.

NOTAS:

61. (acumulador A) − 1 al acumulador A
62. (acumulador B) − 1 al acumulador B
63. (Dirección hexadecimal 4956) − 1 a la dirección hexadecimal 4956
64. (Dirección contenida en el registro índice + 19) − − 1 a la misma dirección

DES": Decrementar puntero de pila (Decrement Stack Pointer)

Se decrementa en 1 el contenido del registro puntero de pila.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
4	34	DES

No se afecta el código de condición.

"DEX": Decrementar registro índice

Se disminuye en 1 el contenido del registro índice.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
4	09	DEX

Sólo se modifica el bit Z del registro de códigos de condición, de acuerdo con la Fig. 8.1.

"EOR": Operación lógica "O exclusiva" (Exclusive OR)

Esta instrucción realiza la operación lógica "O exclusiva" entre cada bit del acumulador A o el acumulador B, y el bit correspondiente de una dirección de memoria. El resultado queda en el acumulador correspondiente. El contenido de la memoria no se modifica.

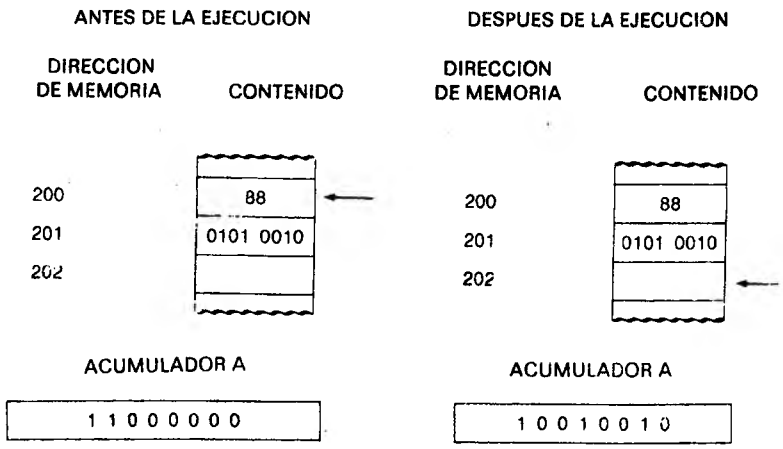
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	88	EOR A #$52	Nota 65 (ver ejemplo)
A DIRECTO	3	98	EOR A $C5	Nota 66
A EXTENDIDO	4	B8	EOR A $53F2	Nota 67
A INDEXADO	5	A8	EOR A $30,X	Nota 68
B INMEDIATO	2	C8	EOR B #$CA	Nota 69
B DIRECTO	3	D8	EOR B $CA	Nota 70
B EXTENDIDO	4	F8	EOR B $CAF2	Nota 71
B INDEXADO	5	E8	EOR B $CA,X	Nota 72

En el registro de códigos de condición se alteran los bits N y Z según la Fig. 8.1 y se borra el bit V.

NOTAS:

65. (acumulador A) + hexadecimal 52; resultado en el acumulador A
66. (acumulador A) + (dirección hexadecimal C5); resultado en el acumulador A
67. (acumulador A) + (dirección hexadecimal 53F2); resultado en el acumulador A
68. (acumulador A) + (dirección especificada por el registro índice + + hexadecimal 30); resultado en el acumulador A
69. (acumulador B) + hexadecimal CA; resultado en el acumulador B
70. (acumulador B) + (dirección hexadecimal CA); resultado en el acumulador B
71. (acumulador B) + (dirección hexadecimal CAF2); resultado en el acumulador B
72. (acumulador B) + (dirección especificada por el registro índice + + hexadecimal CA); resultado en el acumulador B

EJECUCION DE LA INSTRUCCION EOR (EJEMPLO UTILIZANDO EOR A #$52)



"INC": Incrementar

Esta instrucción suma 1 al contenido del acumulador A, del acumulador B o de una posición de memoria.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	4C	INC A	Nota 73
ACUMULADOR B	2	5C	INC B	Nota 74 (ver ejemplo)
DIRECCION EXTENDIDA	6	7C	INC $3C51	Nota 75
DIRECCION INDEXADA	7	6C	INC $26,X	Nota 76

Los bits afectados del registro de códigos de condición son N, Z y V (ver Fig. 8.1).

NOTAS:

73. (acumulador A) + 1; resultado en el acumulador A
74. (acumulador B) + 1; resultado en el acumulador B
75. (Dirección 3C51) + 1; resultado en la misma dirección
76. (Dirección especificada por el registro índice + hexadecimal 26) + 1; resultado en la misma dirección

EJECUCION DE LA INSTRUCCION INC (INC B)

"INS": Incrementar puntero de pila (Increment Stack Pointer)

Esta instrucción suma 1 al contenido puntero de pila.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
4	31	INS

Esta instrucción no afecta ningún bit del registro de códigos de instrucción.

"INX": Incrementar el registro índice

Esta instrucción suma 1 al contenido del registro índice.

	Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
	4	8	INX

Esta instrucción afecta al bit Z del registro de códigos de condición que adopta el valor "1" si los dieciséis bits del resultado son "0", y "0" en caso contrario.

"JMP": Salto incondicional (Jump)

La ejecución de esta instrucción obliga al MP a tomar la instrucción siguiente de una dirección que no es la que sigue en la secuencia.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION EXTENDIDA	3	7E	JMP $54F6	Nota 77
DIRECCION INDEXADA	4	6E	JMP $28,X	Nota 78

Esta instrucción no afecta ningún bit del registro de códigos de condición.

NOTAS:

77. La próxima instrucción a ejecutar se encuentra en la dirección 54F6
78. La próxima instrucción a ejecutar se encuentra en la dirección (registro índice) + hexadecimal 28

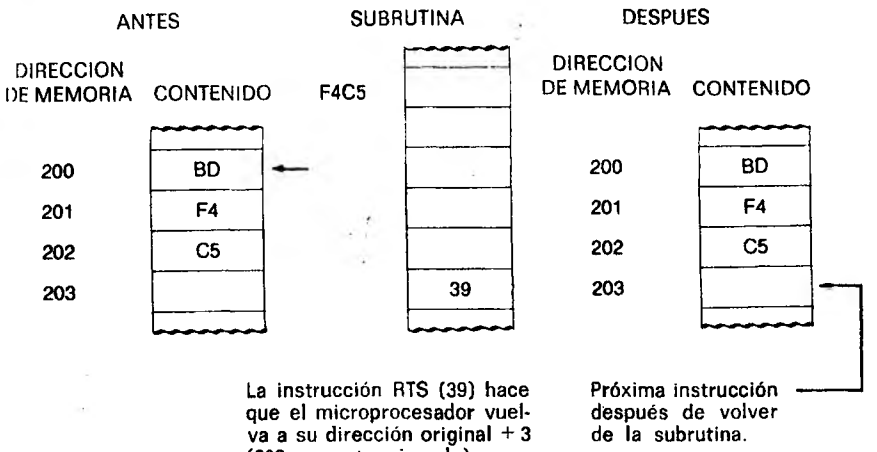
"JSR": Salto incondicional a subrutina (Jump to Subroutine)

Esta instrucción obliga al MP a saltar a una subrutina (otro programa) ubicado en cualquier otra dirección de memoria. La dirección del comienzo de la subrutina se determina en forma similar al caso de la instrucción de salto. Esta instrucción permite que el programador utilice el mismo subprograma (subrutina) en distintas ocasiones durante la ejecución de un programa más grande, en lugar de repetir ese mismo subprograma cada vez que sea necesario. Como final de la subrutina debe haber una instrucción RTS (39) que indica al microprocesador el retorno al programa principal: la dirección original + 3 si el salto a subrutina es en forma extendida, o la dirección original + 2 si se lo hizo en forma indexada.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*
DIRECCION EXTENDIDA	9	BD	JSR $F4C5 (ver ejemplo)
DIRECCION INDEXADA	8	AD	JSR $FF,X

Esta instrucción no modifica los bits del registro de códigos de condición.

EJEMPLO DE INSTRUCCION JSR (JSR $F4C5)



"LDA": Cargar acumulador (Load Accumulator)

Esta instrucción carga el acumulador A o el acumulador B con el contenido de una posición de memoria, el cual no se modifica por la ejecución de la instrucción.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	86	LDA A #$25	Nota 79
A DIRECTO	3	96	LDA A $25	Nota 80
A EXTENDIDO	4	B6	LDA A $2525	Nota 81
A INDEXADO	5	A6	LDA A $25,X	Nota 82
B INMEDIATO	2	C6	LDA B #$4	Nota 83
B DIRECTO	3	D6	LDA B $F2	Nota 84
B EXTENDIDO	4	F6	LDA B $FF41	Nota 85
B INDEXADO	5	E6	LDA B $4,X	Nota 86

Los bits afectados en el registro de códigos de condición son: N y Z (ver Fig. 8.1); V se borra.

NOTAS:

79. Se carga el acumulador A con el número hexadecimal 25
80. Se carga el acumulador A con el contenido de la dirección hexadecimal 25
81. Se carga el acumulador A con el contenido de la dirección hexadecimal 2525
82. Se carga el acumulador A con el contenido de la dirección especificada por el registro índice + hexadecimal 25
83. Se carga el acumulador B con el número hexadecimal 4
84. Se carga el acumulador B con el contenido de la dirección hexadecimal F2
85. Se carga el acumulador B con el contenido de la dirección hexadecimal FF41
86. Se carga el acumulador B con el contenido de la driección especificada por el registro índice + hexadecimal 4

"LDS": Cargar puntero de pila (Load Stack Pointer)

Esta instrucción realiza la carga del registro puntero de pila. El byte más significativo (bits 8 a 15) del puntero de pila se carga desde la dirección especificada por el programa, y el byte menos significativo (0 a 7) se carga desde la dirección especificada en el programa + 1.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION INMEDIATA	3	8E	LDS #$0421	Nota 87
DIRECCION DIRECTA	4	9E	LDS $F2	Nota 88
DIRECCION EXTENDIDA	5	BE	LDS $A421	Nota 89 (ver ejemplo)
DIRECCION INDEXADA	6	AE	LDS $B2,X	Nota 90

Los bits del registro de códigos de condición se modifican de la siguiente manera:

N "1" si la operación de carga colcca en "1" el bit 15 del puntero de pila; "0" en caso contrario.

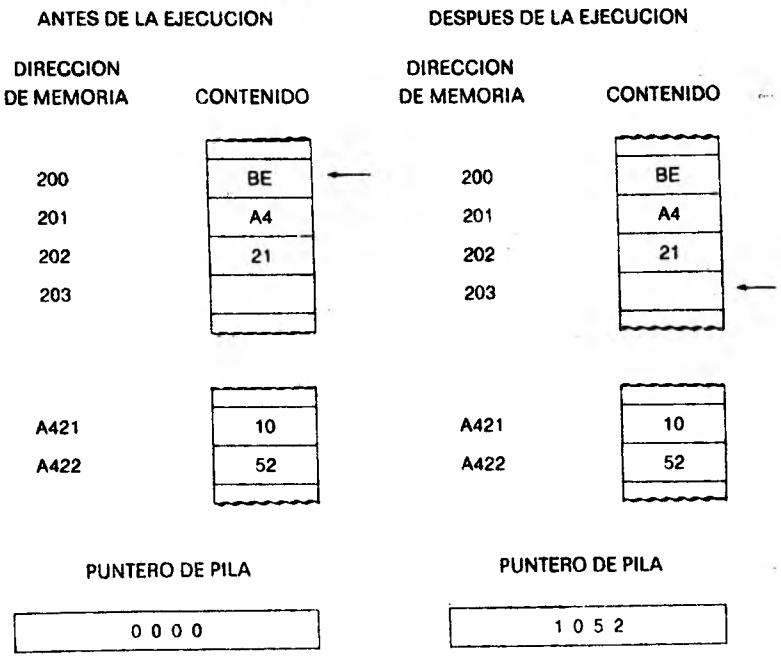
Z "1" si la operación de carga coloca todos los bits del puntero de pila en "0"; "0" en caso contrario.

V Borrado.

NOTAS:

87. Se carga el puntero de pila con el número hexadecimal 0421
88. Se carga el puntero de pila con los contenidos de las direcciones hexadecimales F2 y F3
89. Se carga el puntero de pila con el contenido de las direcciones hexadecimales A421 y A422
90. Se carga el puntero de pila con el contenido de la dirección especificada por el registro índice + B2, y la siguiente

EJEMPLO DE INSTRUCCION LDS (UTILIZANDO LDS $A421)



"LDX": Cargar registro índice (Load index)

Esta instrucción carga el registro índice en forma análoga al caso anterior. El byte más significativo del registro índice se carga desde la dirección especificada en el programa, y el byte menos significativo desde la dirección siguiente a la especificada en el programa.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION INMEDIATA	3	CE	LDX #$3245	Nota 91
DIRECCION DIRECTA	4	DE	LDX $45	Nota 92 (ver ejemplo)
DIRECCION EXTENDIDA	5	FE	LDX $45FE	Nota 93
DIRECCION INDEXADA	6	EE	LDX $98,X	Nota 94

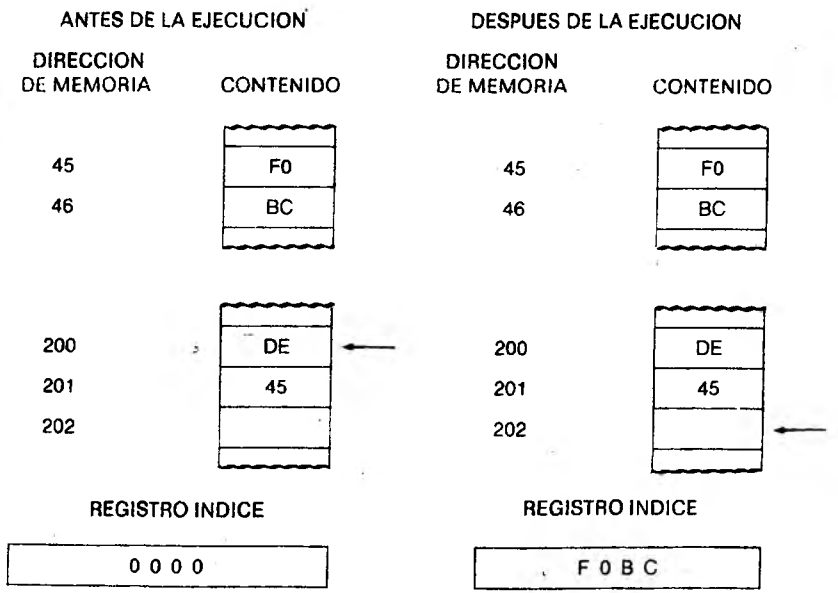
Los bits del registro de códigos de condición se modifican de la siguiente manera:

N "1" si la operación coloca el bit más significativo del registro en "1"; "0" en caso contrario.

Z "1" si la carga del registro coloca "0" en todos sus bits; "0" en caso contrario.

V Borrado.

EJEMPLO DE INSTRUCCION LDX (UTILIZANDO LDX $45)



NOTAS:

91. Se carga el registro índice con el número hexadecimal 3245
92. Se carga el registro índice con los contenidos de las direcciones hexadecimales 45 y 46
93. Se carga el registro índice con los contenidos de las direcciones hexadecimales 45FE y 45FF
94. Se carga el registro índice con los contenidos de la dirección especificada por el registro índice + hexadecimal 98 y la dirección especificada por el registro índice + hexadecimal 99

"LSR": Desplazamiento lógico a la derecha (Logical Shift Right)

Por medio de esta instrucción se desplazan todos los bits del acumulador A, del acumulador B o de una posición de memoria, un lugar a la derecha. El bit 7 se carga con "0" y el bit 0 se desplaza hacia el bit C del registro de códigos de condición.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	44	LSR A	Nota 95
ACUMULADOR B	2	54	LSR B	Nota 96
DIRECCION EXTENDIDA	6	74	LSR $4521	Nota 97
DIRECCION INDEXADA	7	64	LSR $78,X	Nota 98

Los bits del registro de códigos de instrucción se ven afectados de la siguiente manera:

Z "1" si todos los bits del resultado son "0"; "0" en caso contrario.

V "1" si luego de la operación de desplazamiento se cumple alguna de las dos condiciones siguientes: N = 1 y C = 0, ó N = 0 y C = 1; "0" en caso contrario.

C "1" si antes de la operación el bit menos significativo del número a desplazar estaba en "1"; "0" en caso contrario.

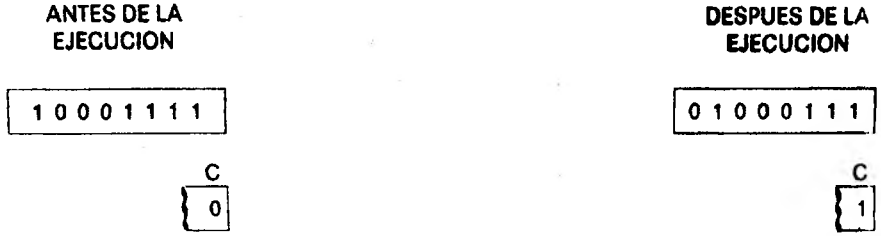
N Borrado.

NOTAS:

95. Desplaza un lugar a la derecha el contenido del acumulador A
96. Desplaza un lugar a la derecha el contenido del acumulador B
97. Desplaza un lugar a la derecha el contenido de la dirección hexadecimal 4521
98. Desplaza un lugar a la derecha el contenido de la dirección especificada por el registro índice + + hexadecimal 78

En todos los casos el bit 7 se carga con 0, y el valor "0" se carga en el bit C del registro de códigos de condición.

EJEMPLO DE DESPLAZAMIENTO (LSR)



"NEG": Negar

Esta instrucción reemplaza el contenido del acumulador A, del acumulador B o de una posición de memoria, con su complemento a 2.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	40	NEG A	Nota 99
ACUMULADOR B	2	50	NEG B	Nota 100
DIRECCION EXTENDIDA	6	70	NEG $2536	Nota 101
DIRECCION INDEXADA	7	60	NEG $C2,X	Nota 102

Los bits del registro de códigos de condición se modifican de acuerdo con lo siguiente:

N, Z Ver Fig. 8.1.

V "1" si se produce un desborde en complemento a 2; esto sólo puede ocurrir si el contenido del acumulador o dirección de memoria a negar es hexadecimal 80.

C "1" si hay un arrastre en la resta realizada. Esto significa que el bit C está en "1" en todos los casos, excepto cuando el contenido del acumulador o dirección de memoria es 0

NOTAS:

99. Se reemplaza el contenido del acumulador A por su complemento a 2
100. Se reemplaza el contenido del acumulador B con su complemento a 2
101. Se reemplaza el contenido de la dirección 2536 con su complemento a 2
102. Se reemplaza el contenido de la dirección indicada por el registro índice + hexadecimal C2 con su complemento a 2

EJEMPLO DE NEGACION

"NOP": No operar

Esta instrucción simplemente hace que el MP avance hacia la dirección de memoria siguiente.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	01	NOP

Esta instrucción no afecta ningún bit del registro de códigos de condición.

"ORA": Operación lógica "O inclusiva" (OR Accumulator)

Esta instrucción ejecuta la operación lógica "O" entre cada bit de alguno de los dos acumuladores y el bit correspondiente de una posición de memoria; el resultado queda en el mismo acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	8A	ORA A #$51	Nota 103
A DIRECTO	3	9A	ORA A $28	Nota 104
A EXTENDIDO	4	BA	ORA A $764A	Nota 105
A INDEXADO	5	AA	ORA A $B6,X	Nota 106
B INMEDIATO	2	CA	ORA B #$78	Nota 107
B DIRECTO	3	DA	ORA B $FA	Nota 108
B EXTENDIDO	4	FA	ORA B $C55D	Nota 109
B INDEXADO	5	EA	ORA B $52,X	Nota 110

Los bits afectados en el registro de códigos de condición son N y Z; V se borra.

NOTAS:

103. (acumulador A) "0" hexadecimal 51
104. (acumulador A) "0" (dirección hexadecimal 28)
105. (acumulador A) "0" (dirección hexadecimal 764A)
106. (acumulador A) "0" (dirección especificada por el registro índice + hexadecimal B6)

Resultado en el acumulador A

107. (acumulador B) "0" hexadecimal 78
108. (acumulador B) "0" (dirección hexadecimal FA)
109. (acumulador B) "0" (dirección hexadecimal C55D)
110. (acumulador B) "0" (dirección especificada por el registro índice + hexadecimal 52)

Resultado en el acumulador B

EJEMPLO DE LA OPERACION "O EXCLUSIVA"

"PSH": Empujar información sobre la pila (Push)

El contenido del acumulador A o del acumulador B se almacenan en memoria en la dirección a la que apunta el registro puntero de pila. Este registro se decrementa luego en 1. El contenido del acumulador afectado no se altera.

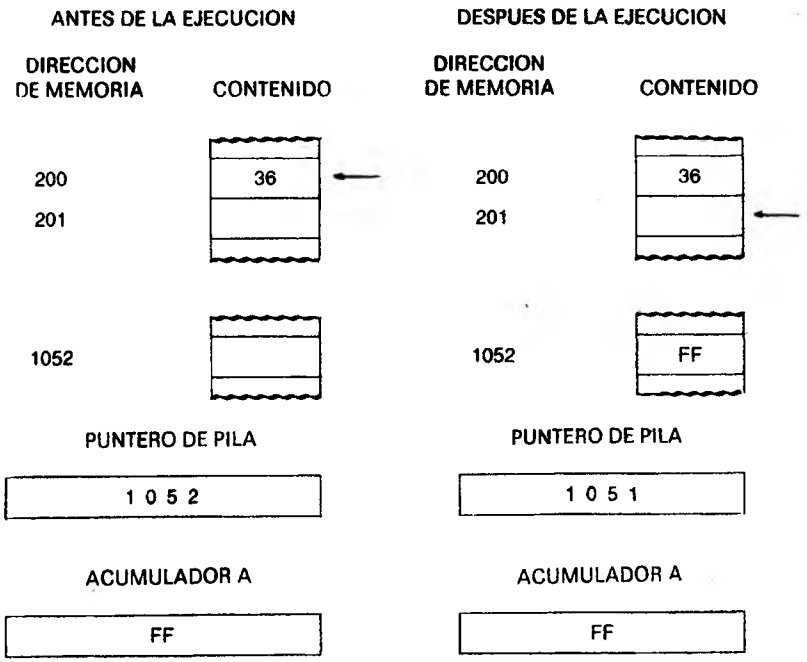
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	4	36	PSH A	Nota 111
ACUMULADOR B	4	37	PSH B	Nota 112

Esta operación no modifica ningún bit del registro de códigos de condición.

NOTAS:

111. El contenido del acumulador A se almacena en la dirección indicada por el puntero de pila; este registro se decrementa en 1
112. El contenido del acumulador B se almacena en la dirección especificada por el puntero de pila; este registro se decrementa en 1

EJEMPLO DE INSTRUCCION PSH A



"PUL": Extraer información de la pila (Pull)

Para ejecutar esta instrucción el registro puntero de pila se incrementa en 1, tras lo cual se descarga en el acumulador A o en el acumulador B el contenido de la dirección a la que apunta el puntero.

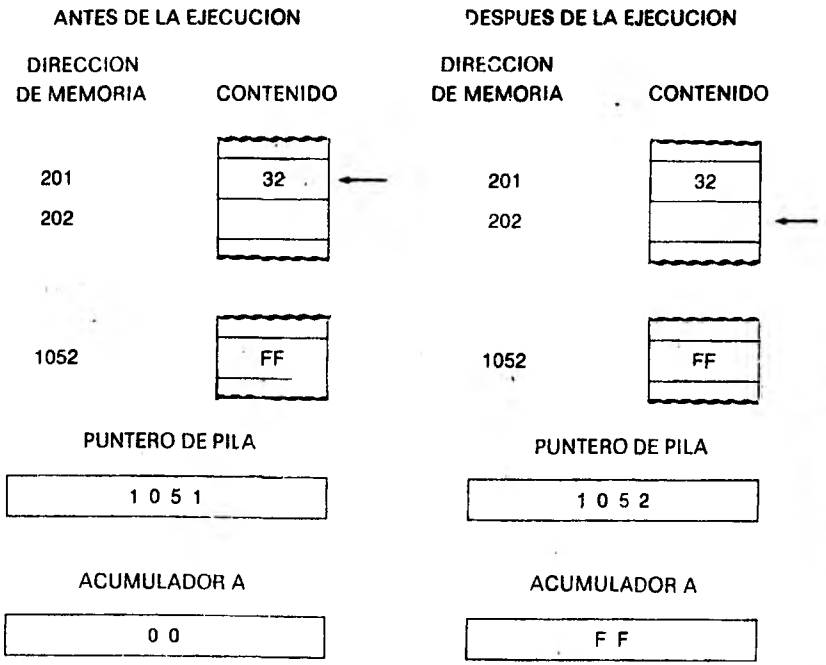
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	4	32	PUL A	Nota 113
ACUMULADOR B	4	33	PUL B	Nota 114

Esta instrucción no afecta ningún bit del registro de códigos de condición.

NOTAS:

113. (Dirección a la que apunta el puntero de pila + 1) cargado en el acumulador A
114. (Dirección a la que apunta el puntero de pila + 1) cargado en el acumulador B

EJEMPLO DE UNA INSTRUCCION PUL A



"ROL": Rotar a la izquierda (Rotate Left)

Esta instrucción desplaza todos los bits del acumulador A, del acumulador B o de una posición de memoria, un lugar hacia la izquierda. El contenido del bit 7 se trasfiere al bit C del registro de códigos de condicion, el que previamente se carga en el bit 0 del operando a desplazar.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	49	ROL A	Nota 115
ACUMULADOR B	2	59	ROL B	Nota 116
DIRECCION EXTENDIDA	6	79	ROL $2426	Nota 117
DIRECCION INDEXADA	7	69	ROL $15,X	Nota 118

Los bits del registro de códigos de condición se alteran según lo siguiente:

N, Z Se modifican según la Fig. 8.1.
V "1" si luego de la operación N y C tienen distinto valor; "0" en caso contrario.
C "1" si antes de la operación el bit más significativo del operando a desplazar vale "1"; "0" en caso contrario.

NOTAS:

115. Rotar a la izquierda un lugar el contenido del acumulador A
116. Rotar a la izquierda un lugar el contenido del acumulador B
117. Rotar a la izquierda un lugar el contenido de la dirección 2426 hexadecimal
118. Rotar a la izquierda un lugar el contenido de la dirección a la que apunta el registro índice + hexadecimal 15

EJEMPLO DE INSTRUCCION ROL A

"ROR": Rotar a la derecha (Rotate Right)

Esta instrucción desplaza todos los bits del acumulador A, del acumulador B o del contenido de una posición de memoria, un lugar a la derecha. El contenido del bit C del registro de códigos de condición se carga en el bit 7 del operando a desplazar, tras lo cual el bit C se carga con el contenido del bit 0 de dicho operando.

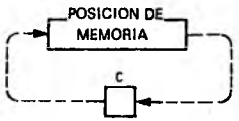
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	46	ROR A	Nota 119
ACUMULADOR B	2	56	ROR B	Nota 120
DIRECCION EXTENDIDA	6	76	ROR $4562	Nota 121
DIRECCION INDEXADA	7	66	ROR $60,X	Nota 122

El registro de códigos de condición se ve modificado según lo siguiente:

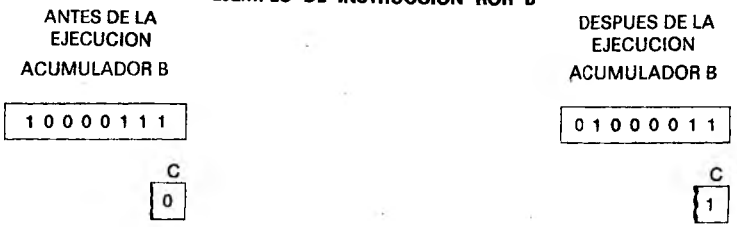
N, Z Ver Fig. 8.1.
V "1" si luego de completar la operación resultan N y C de valores opuestos; "0" en caso contrario.
C "1" si antes de la operación el bit menos significativo del operando a desplazar vale "1"; "0" en caso contrario.

NOTAS:

119. Rotar a la derecha un lugar el contenido del acumulador A
120. Rotar a la derecha un lugar el contenido del acumulador B
121. Rotar a la derecha el contenido de la dirección 4562 hexadecimal
122. Rotar a la derecha un bit el contenido de la dirección a la que apunta el registro índice + hexadecimal 60



EJEMPLO DE INSTRUCCION ROR B



"RTI": Retorno de interrupción (Return from Interrupt)

Cuando el MP reconoce una interrupción, y antes de atenderla, procede a almacenar en la zona de memoria conocida como pila los contenidos del registro de códigos de condición, acumulador A, acumulador B, registro índice y contador de programa. Estos registros y acumuladores se utilizarán con otros fines durante el programa de interrupción. Por lo tanto, si los mismos no hubiesen sido archivados el microprocesador no podría recuperar su estado previo a la interrupción. Se requieren en la zona de almacenamiento temporario siete direcciones de memoria por interrupción; al finalizar el programa de atención de la interrupción debe incluirse una instrucción RTI, que informa al microprocesador que la atención de la interrupción ya se ha completado y puede rescatar de la zona de pila el contenido de los registros y acumuladores, para que los mismos queden en el estado previo a la interrupción.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
10	3B	RTI

"RTS": Retorno de subrutina (Return from subroutine)

Después de ejecutar alguna de las instrucciones BSR o JSR, el MP pasa a ejecutar un programa ubicado en una dirección distinta a la del programa principal. Al completar esta subrutina debe haber una instrucción RTS, que indica al microprocesador que se ha completado la subrutina y debe volver a ejecutar el programa principal, en la dirección original + 2 en el caso de BSR, o en la dirección original + 3 en el caso de JSR.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
5	39	RTS

Esta instrucción no afecta el registro de códigos de condición.

"SBA": Restar acumuladores (Substract Accumulators)

Esta instrucción resta el contenido del acumulador B del contenido del acumulador A, dejando el resultado en el acumulador A; el contenido del acumulador B no se modifica.

	Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
	2	10	SBA

Los bits del registro de códigos de condición afectados por esta instrucción son C, N, Z y V, según la Fig. 8.1.

"SBC": Restar con arrastre (Substract with Carry)

Esta instrucción resta del contenido de uno de los dos acumuladores el contenido de una posición de memoria y el contenido del bit C, dejando el resultado en el mismo acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	82	SBC A #$47	Nota 123
A DIRECTO	3	92	SBC A $F4	Nota 124
A EXTENDIDO	4	B2	SBC A $2588	Nota 125
A INDEXADO	5	A2	SBC A $27,X	Nota 126
B INMEDIATO	2	C2	SBC B #$FA	Nota 127
B DIRECTO	3	D2	SBC B $BB	Nota 128
B EXTENDIDO	4	F2	SBC B $8752	Nota 129
B INDEXADO	5	E2	SBC B $36,X	Nota 130

Los bits afectados en el registro de códigos de condición son:

N, Z, V Según la Fig. 8.1.

C "1" si el valor absoluto del contenido de memoria más el bit de arrastre es mayor que el valor absoluto del contenido del acumulador; "0" en caso contrario.

NOTAS:

123. (acumulador A) – (C) – hexadecimal 47
124. (acumulador A) – (C) – (dirección hexadecimal F4)
125. (acumulador A) – (C) – (dirección hexadecimal 2588)
126. (acumulador A) – (C) – (dirección definida por el registro índice + hexadecimal 27)

} Resultado en el acumulador A

127. (acumulador B) – (C) – hexadecimal FA
128. (acumulador B) – (C) – (dirección hexadecimal BB)
129. (acumulador B) – (C) – (dirección hexadecimal 8752)
130. (acumulador B) – (C) – (dirección definida por el registro índice + hexadecimal 36)

} Resultado en el acumulador B

"SEC": Hacer "1" el bit de arrastre (Set C)

Esta instrucción coloca en "1" el bit C del registro de códigos de condición.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OD	SEC

Esta instrucción afecta el bit C del registro de códigos de condición, que toma el valor "1".

"SEI": Habilitar máscara de interrupciones (Set I)

Esta instrucción coloca en "1" el bit I del registro de códigos de condición. El MP no puede atender interrupciones que ingresen por la línea $\overline{\text{IRQ}}$ hasta que se haya puesto en "0" el bit I. Las interrupciones no enmascarables ($\overline{\text{NMI}}$) no son afectadas.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OF	SEI

En el registro de códigos de condición queda afectado el bit I, que adopta valor "1".

"SEV": Poner en "1" el bit de desborde en complemento a 2 (Set V)

Esta instrucción coloca en "1" el bit de desborde en complemento a 2 (bit V del registro de códigos de condición).

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	OB	SEV

En el registro de códigos de condición se afecta solamente el bit V, que pasa a valer "1".

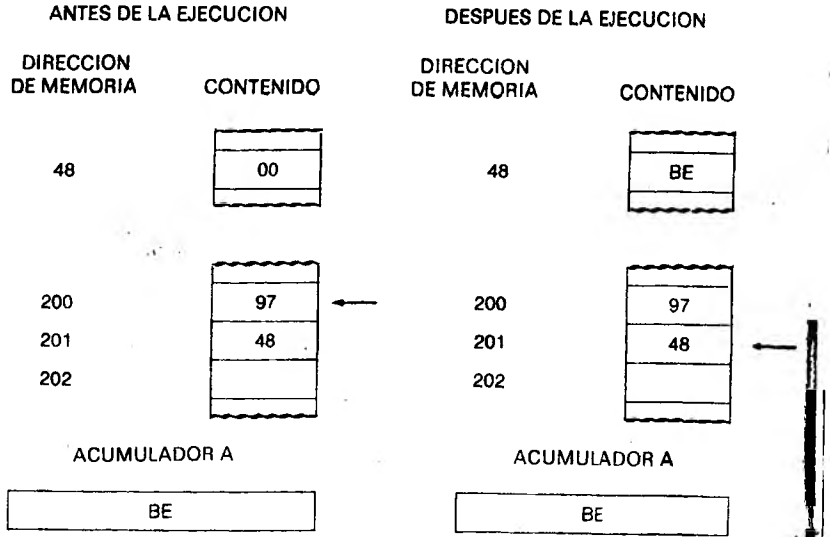
"STA": Almacenar el contenido del acumulador (Store Accumulator)

Esta instrucción almacena el contenido del acumulador A o del acumulador B en una posición de memoria, sin alterar el contenido del acumulador.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A DIRECTO	4	97	STA A $48	Nota 131 (ver ejemplo)
A EXTENDIDO	5	B7	STA A $A268	Nota 132
A INDEXADO	6	A7	STA A $28,X	Nota 133
B DIRECTO	4	D7	STA B $F4	Nota 134
B EXTENDIDO	5	F7	STA B $2652	Nota 135
B INDEXADO	6	E7	STA B $FF,X	Nota 136

Los bits del registro de códigos de condición afectados son N y Z, según la Fig. 8.1; V pasa a valer "0".

EJEMPLO DE INSTRUCCION STA A EN MODO DIRECTO (STA A $48)



NOTAS:

131. Almacenar el contenido del acumulador A en la dirección 48 hexadecimal
132. Almacenar el contenido del acumulador A en la dirección A268 hexadecimal
133. Almacenar el contenido del acumulador A en la dirección a la que apunta el registro índice + 28 hexadecimal
134. Almacenar el contenido del acumulador B en la dirección F4 hexadecimal
135. Almacenar el contenido del acumulador B en la dirección 2652 hexadecimal
136. Almacenar el contenido del acumulador B en la dirección especificada por el registro índice + FF hexadecimal

"STS": Almacenar puntero de pila (Store Stack Pointer)

Esta instrucción almacena en memoria el contenido del puntero de pila. Los ocho bits más significativos del registro se almacenan en la dirección indicada en el programa, en tanto que los ocho bits menos significativos se almacenan en la dirección siguiente.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION DIRECTA	5	9F	STS $B5	Nota 137 (ver ejemplo)
DIRECCION EXTENDIDA	6	BF	STS $26F8	Nota 138
DIRECCION INDEXADA	7	AF	STS $25,X	Nota 139

Los bits afectados en el registro de códigos de condición son:

Z Ver Fig. 8.1.

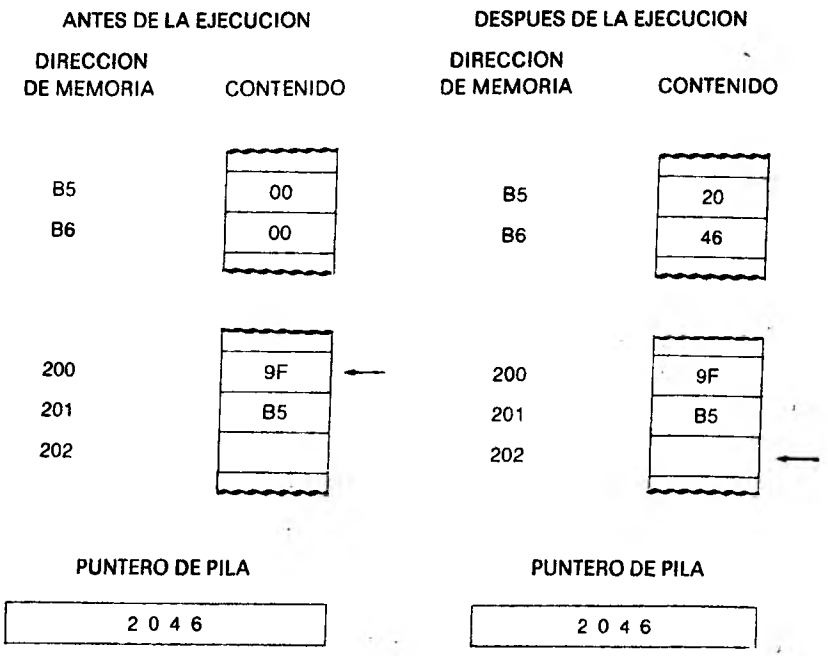
V Borrado.

N "1" si estaba en 1 el bit más significativo (bit 15) del puntero de pila; "0" en caso contrario.

NOTAS:

137. El contenido del puntero de pila se almacena en las direcciones hexadecimales B5 y B6
138. El contenido puntero de pila se almacena en las direcciones hexadecimales 26F8 y 26F9
139. El contenido puntero de pila se almacena en la dirección especificada por el registro índice + hexadecimal 25 y en la dirección especificada por el registro índice + hexadecimal 26

EJEMPLO DE INSTRUCCION STS EN MODO DIRECTO (STS $B5)



"STX": Almacenar registro índice (Store Index)

Esta instrucción almacena en memoria el contenido del registro índice. Los ocho bits más significativos del registro se almacenan en la dirección de memoria especificada en el programa, en tanto que los ocho bits menos significativos se almacenan en la dirección siguiente.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
DIRECCION DIRECTA	5	DF	STX $55	Nota 140
DIRECCION EXTENDIDA	6	FF	STX $326B	Nota 141 (ver ejemplo)
DIRECCION INDEXADA	7	EF	STX $BE,X	Nota 142

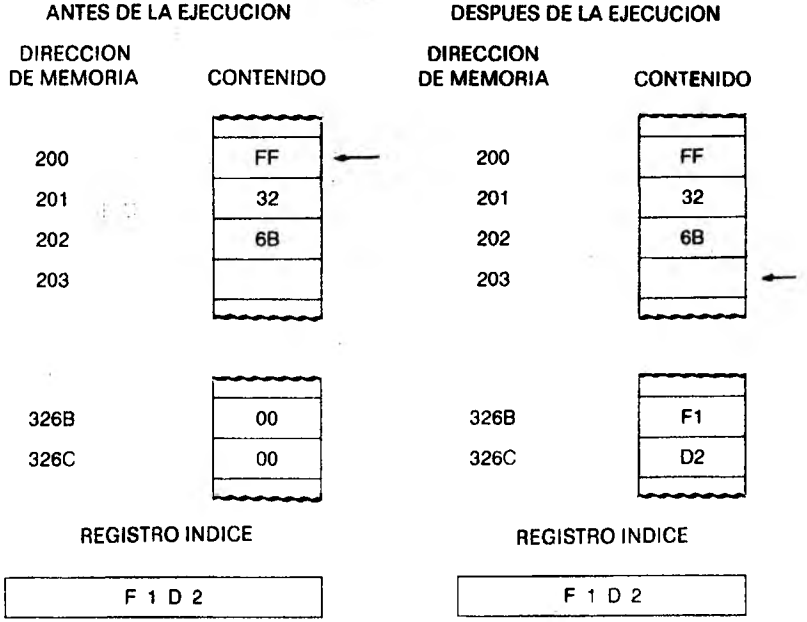
En el registro de códigos de condición se modifican los siguientes bits:

Z Ver Fig. 8.1.
V Borrado.
N "1" si el bit más significativo del registro índice (bit 15) vale 1; "0" en caso contrario.

NOTAS:

140. El contenido del registro índice se almacena en las direcciones hexadecimales 55 y 56
141. El contenido del registro índice se almacena en las direcciones hexadecimales 326B y 326C
142. El contenido del registro índice se almacena en las direcciones especificadas por el mismo registro índice + hexadecimal BE y la siguiente

EJEMPLO DE INSTRUCCION STX EN MODO EXTENDIDO (STX $326B)



"SUB": Restar (Substract)

Esta instrucción resta el contenido de una direccion de memoria del contenido de alguno de los dos acumuladores, dejando el resultado en el mismo acumulador.

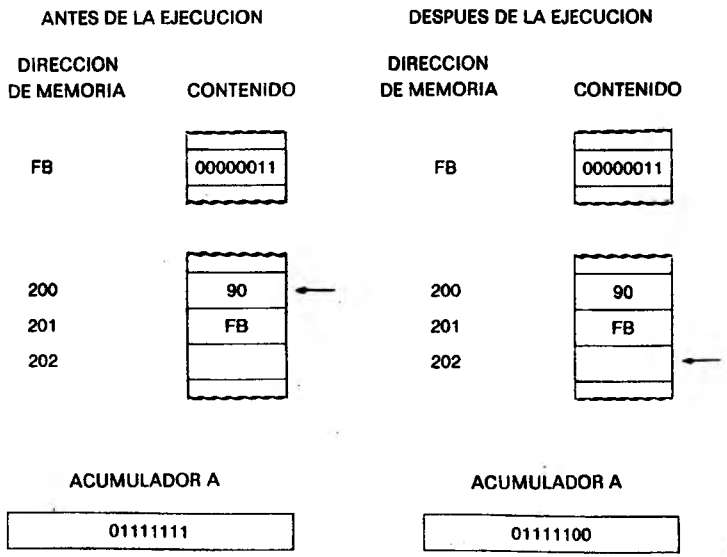
Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
A INMEDIATO	2	80	SUB A #$20	Nota 143
A DIRECTO	3	90	SUB A $FB	Nota 144 (ver ejemplo)
A EXTENDIDO	4	B0	SUB A $1158	Nota 145
A INDEXADO	5	A0	SUB A $15,X	Nota 146
B INMEDIATO	2	C0	SUB B #$59	Nota 147
B DIRECTO	3	D0	SUB B $CB	Nota 148
B EXTENDIDO	4	F0	SUB B $AAC2	Nota 149
B INDEXADO	5	E0	SUB B $FF,X	Nota 150

Los bits afectados en el registro de códigos de condición son:

N, Z, V Según Fig. 8.1.

C "1" si el valor absoluto del contenido de la memoria es mayor que el valor absoluto del contenido del acumulador; "0" en caso contrario.

EJEMPLO DE INSTRUCCION SUB A EN MODO DIRECTO (SUB A $FB)



NOTAS:

143. (acumulador A) — hexadecimal 20
144. (acumulador A) — (dirección hexadecimal FB)
145. (acumulador A) — (dirección hexadecimal 1158)
146. (acumulador A) — (dirección especificada por registro índice + hexadecimal 15)

} Resultado en el acumulador A

147. (acumulador B) — hexadecimal 59
148. (acumulador B) — (dirección hexadecimal CB)
149. (acumulador B) — (dirección hexadecimal AAC2)
150. (acumulador B) — (dirección especificada por registro índice + hexadecimal FF)

} Resultado en el acumulador B

"SWI": Interrupción por programa (Software Interrupt)

Esta instrucción se utiliza cuando se requiere que el MP ejecute algún programa especial. Al detectar esta instrucción el MP almacena en pila los contenidos del registro de códigos de condición, contador de programa, ambos acumuladores y registro índice. Se coloca en "1" la máscara de interrupción (bit I) del registro de códigos de condición, tras lo cual se carga el contador de programa con el contenido de las direcciones FFFA y FFFB, que contienen el puntero de interrupción por programa. El MP ejecuta el programa de interrupción comenzando en la dirección señalada por el puntero mencionado, hasta detectar una instrucción RTI. En ese momento se reponen los valores originales de todos los registros y acumuladores, continuándose el programa en el punto en que se hallaba antes de la instrucción SWI.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
12	3F	SWI

Esta instrucción sólo afecta al bit I del registro de códigos de condición, que adopta el valor "1".

"TAB": Trasferir del acumulador A al acumulador B

Esta instrucción trasfiere el contenido del acumulador A al acumulador B, sin alterar el contenido del acumulador A (se pierde el contenido previo del acumulador B).

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	16	TAB

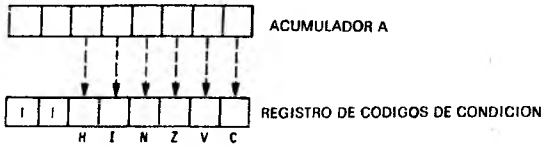
Se modifican los bits N y Z del registro de códigos de condición de acuerdo con la Fig. 8.1, y se borra el bit V.

"TAP": Trasferir del acumulador A al registro de códigos de condición

Esta instrucción trasfiere los contenidos de los bits 0 a 5 del acumulador A a los bits correspondientes del registro de códigos de condición.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	06	TAP

El registro de códigos de condición se modifica en todos sus bits, de acuerdo con los contenidos de los seis bits menos significativos del acumulador A.



C Bit de arrastre (Carry-Borrow)
V Bit de desborde en complemento a 2 (Overflow)
Z Bit de cero
N Bit de negativo
I Máscara de interrupción
H Bit de arrastre intermedio (Half-Carry)

"TBA": Trasferir del acumulador B al acumulador A

Esta instrucción trasfiere el contenido del acumulador B al acumulador A, sin modificar el contenido del acumulador B (se pierde el contenido previo del acumulador A).

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	17	TBA

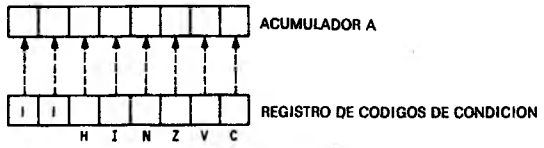
Se modifican los bits N y Z del registro de códigos de condición según la Fig. 8.1 poniendo en 0 el bit V.

"TPA": Trasferir del registro de códigos de condición al acumulador A

Esta instrucción trasfiere el contenido del registro de códigos de condición a los bits 0 a 7 del acumulador A.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
2	07	TPA

No afecta el registro de códigos de condición.



C Bit de arrastre (Carry-Borrow)
V Bit de desborde en complemento a 2 (Overflow)
Z Bit de cero
N Bit de negativo
I Máscara de interrupción
H Bit de arrastre intermedio (Half-Carry)

"TST": Prueba de cero o negativo (Test)

Esta instrucción pone en "1" los bits N y Z del registro de códigos de condición según el valor del contenido del acumulador A, del acumulador B o de una dirección de memoria.

Modo	*Tiempo de ejecución (ciclos)*	*Código de instrucción (hex)*	*Ejemplo*	*Explicación*
ACUMULADOR A	2	4D	TST A	Nota 151
ACUMULADOR B	2	5D	TST B	Nota 152
DIRECCION EXTENDIDA	6	7D	TST $372D	Nota 153
DIRECCION INDEXADA	7	6D	TST $BD,X	Nota 154

Los bits del registro de códigos de condición se modifican de acuerdo a lo siguiente: N y Z según la Fig. 8.1; V y C toman el valor "0".

NOTAS:

151. El contenido del acumulador A define el valor de los bits N y Z
152. El contenido del acumulador B define el valor de los bits N y Z
153. El contenido de la dirección hexadecimal 372D define el valor de los bits N y Z
154. El contenido de la dirección especificada por el registro índice + + hexadecimal BD define el estado de los bits N y Z

"TSX": Trasferir del puntero de pila al registro índice (Transfer from Stack Pointer to Index)

Esta instrucción carga el registro índice con el contenido del puntero de pila más 1. No se modifica el contenido del puntero de pila.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
4	30	TSX

No se modifica ningún bit del registro de códigos de condición.

"TXS": Trasferir del registro índice al puntero de pila (Transfer from Index to Stack Pointer)

Esta instrucción carga el contenido del registro índice menos 1 en el puntero de pila, sin modificar el contenido del registro índice.

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
4	35	TXS

No afecta el registro de códigos de condición.

"WAI": Esperar interrupción (Wait for Interrupt)

Esta instrucción se utiliza cuando se desea que el MP espere algún evento externo; al decodificar esta instrucción se almacenan en pila el registro de códigos de condición, el contador de programa, los acumuladores A y B y el registro índice, tras lo cual se suspende

la ejecución del programa hasta que ocurra una interrupción externa (siempre y cuando el bit I del registro de códigos de condición esté en "0"). Si la interrupción recibida es $\overline{IRQ}$, el bit I adopta el valor "1" y el contador de programa se carga desde las direcciones de memoria FFF8 y FFF9. Si la interrupción es $\overline{NMI}$, tras colocarse en "1" el bit I el contador de programa se carga desde las direcciones de memoria FFFC y FFFD. Estas direcciones contienen los "punteros a los vectores de interrupción", siendo estos vectores las direcciones de comienzo de las respectivas rutinas de atención de la interrupción. El extremo superior de la memoria corresponde a la dirección FFFF, es decir, todas las líneas de dirección en "1".

Como finalización de la rutina de atención de la interrupción se encontrará normalmente la instrucción RTI. Esta instrucción hace que el MP reponga todos los registros con sus contenidos originales, continuando la ejecución del programa principal a partir de la dirección siguiente a la instrucción "WAI".

Tiempo de ejecución (ciclos)	*Código de instrucción (hex)*	*Ejemplo de programa fuente*
9	3E	WAI

En el registro de códigos de condición sólo se modifica el bit I, el que toma el valor "1" tras recibirse la interrupción. Si en el momento de ejecutar la instrucción "WAI" este bit ya estaba en "1", la única forma de salir del estado de espera es por medio de una interrupción no enmascarable $\overline{NMI}$.

8.2 Lenguaje ensamblador del M6800

En el capítulo 6 se analizaron los principios básicos de un ensamblador. En este capítulo se analiza en detalle el ensamblador del M6800. En la sección anterior se han descripto todas las instrucciones que es capaz de manejar el M6800 con todos los modos de direccionamiento correspondientes. Para cada instrucción se señalaron los códigos nemotécnicos y los códigos de operación hexadecimales. El programa ensamblador traduce los *programas fuente*, escritos en código nemotécnico, a *programas objeto*, esto es, programas escritos en lenguaje de máquina. Para que el ensamblador pueda "entender" lo que se pretende ejecutar en un programa, deben tenerse en cuenta una serie de reglas relacionadas con la escritura del programa fuente.

Debe recordarse que el ensamblador no es más que un programa, en particular, un programa escrito para hacer más sencillo el trabajo del programador. Es mucho más simple escribir un programa fuente usando códigos nemotécnicos, que escribir un programa directamente en lenguaje de máquina. Esta sección permite ilustrar cómo deben escribirse los programas fuente, que posteriormente se ensamblarán en los distintos sistemas sobre los cuales pueda ejecutarse el ensamblador del M6800. En algunos casos, de acuerdo con el sistema utilizado para el ensamble pueden presentarse ligeras diferencias.

Número de línea

En la Sección 8.1 se representó un ejemplo de listado para cada instrucción. Por ejemplo, LDA A #42 es una sentencia de programa fuente. El programa fuente debe contener *una única* sentencia por línea. Cada línea debe comenzar por un *número de línea* (comúnmente llamado número de secuencia), y cada número de línea debe ser mayor que el número de la línea anterior. Los números de línea pueden tener de 1 a 5 dígitos; no tiene importancia cuál es el primer número de línea, y no es necesario que sean consecutivos. El ejemplo siguiente muestra números de línea aceptables:

```
100  Sentencia Nº 1
102  Sentencia Nº 2
110  Sentencia Nº 3
130  Sentencia Nº 4
```

Como puede verse, los únicos requisitos existentes son que cada número de línea sea mayor que el anterior, y que tenga entre uno y cinco dígitos de longitud. Debe enfatizarse que los números de línea sólo son para uso del programa ensamblador y *no* forman parte del programa.

Rótulos

Al analizar el modo de direccionamiento relativo en el capítulo 7, se consideraron las instrucciones de bifurcación o salto, es decir, la trasferencia a una dirección que no está en secuencia con la que se acababa de ejecutar. Si se escriben programas en lenguaje fuente, el ensamblador hace el cálculo de la trasferencia, para lo cual deben definirse *grupos de caracteres alfanuméricos* que se denominan "rótulos", los cuales deben seguir al número de línea dejando un blanco de por medio. Cuando este mismo rótulo, por ejemplo "TEMP", aparece como parte de una sentencia fuente en algún lado del programa, por ejemplo "BRA TEMP", el ensamblador calcula el segundo byte de la instrucción de bifurcación. El ejemplo siguiente señala cómo debe utilizarse un rótulo:

```
100  TEMP LDA A #41
101       .    .
 .        .    .
 .        .    .
 .        .    .
125       BRA  TEMP
```

La asignación de rótulos generalmente se hace de manera que signifiquen algo para el programador, aunque deben observarse algunas reglas:

1. El rótulo debe tener entre uno y seis caracteres alfanuméricos.
2. El primer carácter debe ser alfabético.
3. Debe comenzar dejando un espacio luego del número de línea.
4. Un rótulo puede utilizarse una sola vez siguiendo a un número de línea. Puede usarse en más de una sentencia fuente (BRA TEMP, JMP TEMP, etc.).
5. No puede usarse como rótulo ninguno de los caracteres A, B ó X por separado.

Algunos ejemplos de rótulos son: TEMP1, TEMP2, COINA, HLP21, HLP22, STP5A, TBL1, TOM1, DICKA, etc.

Sentencias fuente

Las sentencias del programa fuente deben empezar en el tercer lugar (o cualquiera posterior al tercero) luego de un número de línea, o en el segundo lugar (o en cualquiera posterior al segundo) luego de un rótulo. Ejemplo:

```
100 TEMP STA A $52
    ↑
    un espacio con rótulo

100  STA A $52
     ↑
     dos espacios sin rótulo
```

A medida que se escriben las sentencias del programa fuente, debe indicarse el modo de direccionamiento que se está utilizando, y el sistema de numeración en el que se expresan las cantidades involucradas. El signo numeral indica el modo inmediato, es decir, el operando se encuentra en el byte siguiente al código de operación.

La falta del signo numeral indica alguno de los otros modos de direccionamiento. El ensamblador determina el modo analizando la instrucción y el operando de la sentencia fuente:

#	indica modo de direccionamiento inmediato
$	indica operando hexadecimal
@	indica número en octal
%	indica número en binario
'	indica un carácter en código ASCII

NOTA: Si un número del programa fuente no lleva ninguno de los símbolos indicados, el programa ensamblador lo considera un número decimal.

A continuación se indican ejemplos de sentencias fuente:

1. Para almacenar el contenido del acumulador A en la dirección hexadecimal 52:

```
100 TEMP STA A $52
    ↑    ↑   ↑ ↑
    un solo espacio
```

2. Para almacenar el contenido del acumulador A en una dirección a la que se le ha asignado el rótulo TEMP:

```
100  STA A TEMP
  ↑    ↑ ↑
         un solo espacio
     doble espacio
```

3. Para almacenar el número binario 11000111 en el acumulador A:

```
100  LDA A #%11000111
  ↑    ↑ ↑
         un solo espacio
     doble espacio
```

Cada una de las sentencias anteriores será convertida a su equivalente en lenguaje de máquina y ubicada en direcciones consecutivas de memoria.

Comentarios

Muchas veces, mientras se escribe un programa, resulta conveniente insertar comentarios que clarifiquen lo que se está haciendo. Si luego de un número de línea, y dejando un blanco, se coloca un asterisco, el ensamblador ignora el resto de la línea, aunque la imprime en el listado del programa ensamblado:


```
100 * ESTA RUTINA CALCULA VELOCIDAD
    ↑
    un solo espacio
```

Comentarios de este estilo pueden ser de ayuda considerable para el programador (o para alguna otra persona) al analizar la salida ensamblada. Pueden colocarse comentarios luego de una sentencia fuente y en la misma línea, si se los separa de aquélla con uno o dos espacios.

Directivas para el ensamblador

De la misma manera que se ha presentado una directiva para especificar el sistema numérico en el que se representa una cantidad, pueden usarse otras directivas para su interpretación por parte del programa ensamblador.

1. *ORG (Origen):* La *sentencia ORG* indica al ensamblador cuál es la dirección de comienzo del programa a ensamblar. Si no se incluye sentencia ORG, el programa ensamblador coloca la primera instrucción del programa a ensamblar en la dirección 0. Por ejemplo, si se dan las sentencias,

```
100  ORG $425
105  LDA A #$63
   ↑
   dos espacios
```

el programa ensamblador asignará al código de operación de la instrucción LDA A la dirección hexadecimal de memoria 0425, asignando al operando hexadecimal 63 la dirección de memoria 0426. Cada vez que en el programa aparezca una sentencia ORG, el ensamblador asigna las direcciones a partir de la indicada en esa sentencia.

2. *EQU (Asignar valor a un símbolo):* Suele ser conveniente asignar valor a un símbolo (rótulo) de un programa. Por ejemplo: si se desea asignar al símbolo PIA1AC el valor hexadecimal 4005, se utiliza una *sentencia EQU* antes de la sentencia ORG, según se indica:

```
100 PIA1AC EQU $4005
105  ORG $500
110  LDA A PIA1AC
```

Esto permite utilizar el símbolo (rótulo) PIA1AC en cualquier lugar del listado fuente, aunque la salida del ensamblador incluye el valor real del rótulo. Por ejemplo, para la sentencia fuente LDA A PIA1AC la salida ensamblada es:

Dirección de memoria	*Contenido*	*Explicación*
500	10110110 (B6)	LDA A (Modo extendido)
501	01000000 (40)	($4005)
502	00000101 (05)	

3. *RMB (Reservar byte en memoria):* La directiva RMB se utiliza para reservar espacio de memoria para rótulos. Se coloca *después* de la sentencia ORG, dado que el ensamblador debe asignar al espacio de memoria una dirección que queda justamente determinada por la sentencia ORG. Es común el uso de rótulos asociados con la directiva RMB. Por ejemplo:

```
100  ORG $200
101 TEMP RMB 1
```

Esto hace que el ensamblador asigne el rótulo TEMP a la dirección hexadecimal 200. Si la línea fuese 101 TEMP RMB 3, se asignaría la dirección hexadecimal 200 al rótulo TEMP, la dirección 201 al rótulo TEMP + 1, y la dirección 202 al rótulo TEMP + 2; esto implica haber reservado esos bytes de la memoria para el rótulo TEMP y sus consecutivos TEMP + 1 y TEMP + 2. El programador puede hacer referencia a estas direcciones utilizando los rótulos, en lugar de tener que recordar los valores numéricos reales. Ejemplos de sentencias fuente que utilizan los rótulos así definidos:

```
150  LDA B TEMP
155  STA A TEMP + 2
160  STA B TEMP + 1
```

4. *NAM (Nombre):* La directiva NAM asigna un nombre al programa. Debe colocarse en el tercer espacio luego del número de línea, y debe estar seguido de un nombre de programa de hasta ocho caracteres. Por ejemplo:

```
100  NAM RONBI
      ↑
   doble espacio
```

La directiva NAM generalmente es la primera sentencia del programa fuente.

5. *SPC (Espacio):* Esta directiva produce lugares verticales en blanco durante el listado de la salida ensamblada. Por ejemplo:


```
190  LDA A #52
200  SPC 2
210  STA A $FF
```

Como resultado de esta directiva se obtienen dos líneas en blanco entre la salida de código de máquina correspondiente a la línea 190 y la correspondiente a la línea 210 en el listado ensamblado.

6. *FCB (Formar constante de un byte):* Esta directiva asigna valores constantes a direcciones consecutivas de memoria, por ejemplo:

```
200  ORG $400
205 TABLE FCB 0, $25, $FE, $40
```

En la salida ensamblada se obtiene la siguiente asignación de posiciones de memoria:

Dirección de memoria	*Contenido hexadecimal*
400	00
401	25
402	FE
403	40

7. *FDB (Formar constante de dos bytes):* Esta directiva es similar a la FCB, con la diferencia de formar constantes que ocupan dos bytes de memoria. Por ejemplo:

```
200  ORG $400
206 TABLE FDB 0, $BF51, $55C1
```

La salida ensamblada asigna los números de dos bytes a sucesivas direcciones de memoria, en la forma siguiente:

Dirección de memoria	*Contenido hexadecimal*
400	00
401	00
402	BF
403	51
404	55
405	C1

8. *FCC (Formar constante de un carácter):* Esta directiva se utiliza para almacenar caracteres ASCII en direcciones de memoria consecutivas. Por ejemplo, el código ASCII para la letra P es 1010000

(50_{16}), que se almacena en memoria como 01010000. La directiva FCC permite que el programador liste caracteres ASCII en su programa fuente. El ensamblador convierte estos caracteres a su equivalente binario ASCII, asignando el número binario a la dirección de memoria adecuada de acuerdo con las sentencias ORG. Por ejemplo:

```
200 ORG $500
210 TABLE FCC "ALL GREAT!"
```

El ensamblador asigna el código binario ASCII equivalente a la expresión mencionada, en la forma siguiente:

Carácter	*Dirección*	*Contenido*
A	500	01000001
L	501	01001100
L	502	01001100
blanco	503	00100000
G	504	01000111
R	505	01010010
E	506	01000101
A	507	01000001
T	508	01010100
!	509	00100001

NOTA: El delimitador (" " en este ejemplo) debe ser un único carácter no numérico. Esto significa que [ ] no es legal, dado que son dos caracteres diferentes, aunque [[sí sería válido.

9. *MON (Monitor):* Esta directiva se utiliza para indicarle al ensamblador el fin del programa fuente. Tras ensamblar el programa, se retorna el control al monitor (terminal) del sistema. Por ejemplo:

```
200  MON
     ↑
     dos espacios
```

No puede utilizarse rótulo con la directiva MON.

10. *END (Fin):* Esta directiva se utiliza para ensamblar más de un programa fuente; luego de ensamblar un programa que termina en una directiva END, el ensamblador pasa a ensamblar otro programa fuente. Se devuelve el control al sistema operativo sólo cuando se encuentra una directiva MON. Por ejemplo:

```
201  END
     ↑
     dos espacios
```


No debe usarse rótulo con esta directiva.

11. *PAGE (Página):* Esta directiva hace que el ensamblador desplaze el papel del sistema de impresión hasta el comienzo de la página siguiente. Esto significa que después de esta directiva el listado continúa al comienzo de la página siguiente. Por ejemplo:

```
205  PAGE
     ↑
     dos espacios
```

No debe utilizarse rótulo con esta directiva.

12. *OPT (Opciones de salida):* Esta directiva provee formatos opcionales para la salida ensamblada. Pueden introducirse distintas opciones en una misma línea, separándolas mediante una coma. Las opciones disponibles son las siguientes:

Descripción	*Listado en el programa fuente*
a) Generar una cinta con el programa objeto. Se supone seleccionada la opción a menos que se indique lo contrario	OPT O
b) No generar cinta objeto	OPT NOO
c) Almacenar el programa objeto en la memoria del sistema	OPT M
d) No almacenar el programa objeto	OPT NOM
e) Imprimir todos los símbolos (rótulos) al final del ensamble	OPT S
f) No imprimir símbolos	OPT NOS
g) Imprimir el listado de la salida ensamblada	OPT L
h) No imprimir el listado de la salida ensamblada	OPT NOL
i) No imprimir encabezamiento al comienzo de cada página. Se imprimirá el encabezamiento si no se utiliza esta directiva	OPT NOP
j) Imprimir toda la información generada por las directivas FCC, FCB y FDB. Esta directiva se supone implícita si no se especifica la directiva contraria	OPT G
k) Imprimir sólo la primera línea de los datos generados por las directivas FCC, FCB y FDB	OPT NOG

Ejemplo de uso de directivas:

```
200  OPT M,S
     ↑
     dos espacios
```

8.3 Ejemplo de programa fuente

Se resumirá a continuación la forma de escribir un programa fuente para traducirlo al lenguaje de máquina mediante un ensamblador.* La siguiente sección de programa se presenta para ilustrar la mayor cantidad posible de las reglas recién enumeradas:

```
100  NAM RONB
105  OPT M, S, L
110 PIA1AC EQU $4005
115 TEMP EQU $251
120  ORG $10
125 COIN RMB 5
130  ORG $20
135 TAB FCB $10,$20,$30
140  ORG $30
145 TAB1 FDB $1000,$2000,$3000
150  PAGE
155  ORG $60
160 TYPE FCC *HELP*
165  SPC 6
170 *"COMIENZO DEL PROGRAMA PRINCIPAL"
175  ORG $300
180 NEW LDA A PIA1AC
185  STA A COIN
190 AGAIN LDA B TAB1+1
195  STA B COIN+1
 .    .
 .    .
220  BRA NEW
 .    .
 .    .
300  END
305  MON
```

* Aquí y en lo sucesivo, a menos que se indique lo contrario se hace referencia al ensamblador del M6800. Las reglas pueden ser distintas para otros ensambladores. (*N. del T.*)

Nótese que todas las sentencias del programa fuente, excepto los rótulos y comentarios, tienen dos espacios entre el número de línea y la sentencia. Los rótulos y los comentarios sólo requieren un espacio luego del número de línea.

Se analiza a continuación cada línea del programa:

100 NAM RONB: Asigna RONB como nombre del programa. Este nombre se imprimirá en el encabezamiento del listado de la salida ensamblada.

105 OPT M, S, L: Estas opciones hacen que el ensamblador almacene el código en memoria (M), que imprima la tabla de símbolos (S) y que imprima el programa ensamblado (L).

110 PIA1AC EQU $4005: Se asigna el valor hexadecimal 4005 al rótulo PIA1AC.

115 TEMP EQU $251: Se asigna el valor hexadecimal 0251 al rótulo TEMP.

120 $10 y *125 COIN RMB 5:* Estas dos líneas harán que el ensamblador asigne el rótulo COIN a la dirección hexadecimal 10, el rótulo COIN + 1 a la dirección 11, etc. Estos rótulos sólo se usan en el programa fuente.

130 ORG $20 y *135 TAB FCB $10,$20,$30:* Estas dos líneas asignan el rótulo TAB a la dirección hexadecimal 20. En la dirección 20 se carga el valor hexadecimal 10, en la dirección 21 se carga el valor hexadecimal 20 y en la dirección 22 el valor hexadecimal 30.

140 ORG $30 y *145 TAB1 FDB $1000,$2000,$3000:* Estas dos líneas asignan el rótulo TAB1 a la dirección hexadecimal 30 y cargan la memoria de acuerdo con la siguiente tabla:

Dirección	*Dato (hexadecimal)*
30	10
31	00
32	20
33	00
34	30
35	00

150 PAGE: El programa ensamblador saltea el resto de la página y comienza a imprimir en la primera línea de la página siguiente.

155 ORG $60 y 160 TYPR FCC "HELP": Con estas dos líneas se asigna el rótulo TYPE a la dirección hexadecimal 60 y se almacenan los valores binarios correspondientes a los códigos ASCII de la palabra HELP en direcciones de memorias consecutivas comenzando en la dirección 60, según se indica:

Dirección	*Dato hexadecimal (ASCII)*	
60	48	H
61	45	E
62	4C	L
63	50	P

165 SPC 6: El ensamblador deja seis espacios en blanco.

170 "COMIENZO DEL PROGRAMA PRINCIPAL": Se imprime esta línea en el listado de salida.

175 ORG $300 y 180 NEW LDAA PIA1AC: Se asigna el rótulo NEW a la dirección hexadecimal 300 y el código de la instrucción LDA A PIA1AC a las direcciones 300, 301 y 302 según se indica:

Dirección	*Contenido (hexadecimal)*	
300	B6	
301	40	(En la línea 110 se asignó el
302	05	valor 4005 al rótulo PIA1AC)

185 STA A COIN: Se asigna el código de la instrucción STA A (modo de direccionamiento directo) a las direcciones hexadecimales 303 y 304 según se indica:

Dirección	*Contenido (hexadecimal)*	
303	97	(En la línea 125 se asignó el
304	10	valor 10 al rótulo COIN)

190 AGAIN LDAB TAB1 + 1: Se asigna el rótulo AGAIN a la dirección hexadecimal 305, ubicando el código binario de la instrucción en las direcciones 305 y 306 según se indica:

Dirección	*Contenido (hexadecimal)*	
305	D6	(En la línea 145 se asignaron
306	31	los valores 30, 31, etc., a los rótulos TAB1, TAB1 + 1, etc.)

195 STA B COIN + 1: Se asigna el código de máquina de la instrucción a las direcciones hexadecimales 307 y 308 según se indica:

Dirección	*Contenido (hexadecimal)*	
307	D7	(En la línea 125 se asignó el
308	11	valor 11 a COIN + 1)

220 BRA NEW: El ensamblador debe calcular el número de posiciones de memoria a saltar para encontrar la próxima instrucción. La salida ensamblada es la siguiente:

Dirección	*Contenido (hexadecimal)*
309	20
30A	F5

El valor F5 es el complemento a 2 del valor hexadecimal OB (dirección actual + 2 [30B] dirección del rótulo NEW [300]).

300 END: Se indica al ensamblador el final del programa.

305 MON: Se devuelve el control al monitor.

En el capítulo 11 se encontrarán ejemplos de salidas ensambladas de distintos programas.

8.4 Ejemplos de instrucciones de bifurcación (Branch)

Un programa que se ejecuta en la memoria de un MP encuentra a menudo instrucciones de bifurcación (Branch). En esta sección se explica cómo determina el MP si al encontrar una instrucción de este tipo debe bifurcar o continuar con la próxima instrucción en la secuencia. Debe recordarse que, salvo en los casos de instrucciones BSR (bifurcación a subrutina) o BRA (bifurcación incondicional), la deci-

sión depende del estado de determinados bits del registro de códigos de condición (estas bifurcaciones se conocen como "bifurcaciones condicionales"). Estos bits quedan en "0" ó "1" luego de ejecutarse alguna instrucción previa a la posible bifurcación; ese estado debe ser favorable para que la misma pueda realizarse. Si se realiza la bifurcación se obtiene la dirección de destino de acuerdo con lo visto en la Sección 7.8; en caso contrario el MP ejecuta la *próxima instrucción en la secuencia* (dirección actual + 2). A continuación se dan algunos ejemplos:

	INSTRUCCIONES	ESTADO DEL REGISTRO DE CODIGOS DE CONDICION LUEGO DE LA EJECUCION						¿OCURRE LA BIFURCACION CONDICIONAL?
		H	I	N	Z	V	C	
1	LDA B #$FF	NA	NA	1	0	0	NA	
	TST B	NA	NA	1	0	0	0	
	BCC	NA	NA	1	0	0	0	Sí
2.	LDA B #$FF	NA	NA	1	0	0	NA	
	COM B	NA	NA	0	1	0	1	
	BCC	NA	NA	0	1	0	1	No
3.	LDA A #0	NA	NA	0	1	0	NA	
	ADD A #$80	0	NA	1	0	0	0	
	BHI	0	NA	1	0	0	0	Sí
4.	LDA A #01	NA	NA	0	0	0	NA	
	ASR A	NA	NA	0	1	1	1	
	BLE	NA	NA	0	1	1	1	Sí
5.	LDA A #$85	NA	NA	1	0	0	NA	
	LDA B #$05	NA	NA	0	0	0	NA	
	SBA	NA	NA	1	0	0	0	
	BMI	NA	NA	1	0	0	0	Sí
6.	LDX #$01	NA	NA	0	0	0	NA	
	DEX	NA	NA	0	1	0	NA	
	BEQ	NA	NA	0	1	0	NA	Sí
7.	LDA A #7F	NA	NA	0	0	0	NA	
	ROR A	NA	NA	0	0	1	1	
	BVS	NA	NA	0	0	1	1	Sí
8.	LDA A #7F	NA	NA	0	0	0	NA	
	ADD A #1	NA	NA	1	0	1	0	
	COM A	NA	NA	0	0	0	1	
	BGE	NA	NA	0	0	0	1	Sí
9.	LDA A #7F	NA	NA	0	0	0	NA	
	LDA B #7E	NA	NA	0	0	0	NA	
	SBA	NA	NA	0	0	0	0	
	LSR A	NA	NA	0	1	1	1	
	BCS	NA	NA	0	1	1	1	Sí

\# Indica modo inmediato.
$ Indica número hexadecimal
NA Indica bit no afectado.

Problemas*

1. Escribir las instrucciones de lenguaje fuente para el M6800 para realizar las siguientes operaciones:
   a) Almacenar el contenido del acumulador B en la dirección hexadecimal 2452.
   b) Almacenar el contenido del acumulador B en la dirección hexadecimal 24.
   c) Almacenar el contenido del acumulador B en la dirección que se obtiene sumándole hexadecimal 24 al contenido del registro índice.
   d) Decrementar el acumulador A.
   e) Cargar el acumulador A con el contenido de la dirección hexadecimal 24.
   f) Sumar 16 al contenido del acumulador B.
   g) Comparar el contenido del registro índice con el de la dirección de memoria ABFD.
   h) Cargar el número hexadecimal CD52 en el registro índice.
2. ¿Cuál es el estado de cada bit del registro de códigos de condición después de ejecutar las instrucciones del problema 1?
3. Representar cada una de las instrucciones del problema 1 por medio de los códigos binarios (lenguaje de máquina) almacenados en memoria.
4. Calcular el desplazamiento (valor que debe sumarse a la dirección de memoria de la instrucción de bifurcación + 2) para las siguientes bifurcaciones:

	Dirección	*Instrucción*	
a)	500	20	(Bifurcación incondicional
	501	—	a la dirección 520)
b)	500	20	(Bifurcación incondicional
	501	—	a la dirección 491)

* X indica el registro índice, PC el contador de programa (Program Counter), SP el puntero de pila (Stack Pointer) y CC el registro de códigos de condición (*N. del T.*)

5. Examinar el programa siguiente comenzando en la dirección 00F9. Al *completar* la ejecución de la instrucción que termina en la dirección 106, indicar los contenidos de cada uno de los registros del MP:

	Dirección hexadecimal	*Contenido hexadecimal*
	00F9	D6
	00FA	FE
A _____	00FB	86
	00FC	AB
B _____	00FD	FE
PC _____	00FE	01
	00FF	02
X _____	0100	20
SP _____	0101	02
	0102	00
H I N Z V C	0103	FA
CC [1 1 X _ _ _ _]	0104	AE
	0105	02
	0106	1B

6. Expresar la instrucción LDA A en cuatro modos de direccionamiento (definiendo arbitrariamente los valores de los datos, direcciones y desplazamientos):

	LDA A	
Inmediato	_____	_____
Directo	_____	_____
Extendido	_____	_____
Indexado	_____	_____

7. Analizar cada uno de los programas siguientes, comenzando en la primera dirección indicada. Indicar los contenidos de cada registro al completarse el programa.

a)

A _____
B _____
PC _____
X _____
SP _____

Posición de memoria	*Contenido*
FA	CE
FB	01
FC	03
FD	E6
FE	00
FF	20
100	03
101	01
102	3F
103	B2
104	B6
105	01
106	01
107	9E
108	FB

b)

A _____
B _____
PC _____
X _____
SP _____

Posición de memoria	*Contenido*
F9	FE
FA	01
FB	00
FC	AE
FD	00
FE	20
FF	02
100	01
101	03
102	D6
103	FC
104	86
105	13

c)

A _____

B _____

PC _____

X _____

SP _____

X contiene inicialmente 00FB

Posición de memoria	*Contenido*
F8	EE
F9	02
FA	F6
FB	01
FC	03
FD	96
FE	F9
FF	8E
100	20
101	A1
102	4A
103	26
104	FD

CAPITULO

9

Familia del microcomputador M6800

En el capítulo 8 se analizó el juego completo de instrucciones del M6800. Antes de poder utilizar estas instrucciones debe contarse con una estructura circuital que sea capaz de implementarlas y ejecutarlas. Este capítulo analiza la familia de circuitos integrados asociada con el microprocesador M6800, describiéndose cada uno de los dispositivos, incluyendo todos los registros internos, los números de terminal correspondientes y todas las señales aplicables a dichos terminales.

9.1 Descripción general del sistema

La familia M6800 de circuitos integrados LSI (*L*arge Scale *Int*egration - integración en gran escala) permite el diseño de un sistema con mínimo esfuerzo y mínimo tiempo de diseño. La unidad central de procesamiento M6800 es el núcleo de la familia, y entre los integrados de apoyo básico se cuentan:

1. *Memoria de acceso al azar* (Random Access Memory - RAM) MCM6810
2. *Adaptador de periféricos* (Peripheral Interface Adapter - PIA) MC6821
3. *Memoria de lectura solamente* (Read Only Memory - ROM) MCM6830
4. *Adaptador asincrónico de comunicaciones* (Asynchronous Communications interface Adapter - ACIA) MC6850

El MP selecciona estos dispositivos a través de una "barra de direcciones" de 16 bits, realizándose la trasferencia de información y datos a través de una "barra de datos" de 8 bits.

INTEGRACION EN GRAN ESCALA (LSI): Este es un término que se aplica a pastillas (o circuitos integrados) que contienen muchos circuitos electrónicos. El número necesario para justificar la denominación LSI está sujeto a discusión, aunque números superiores a 1000 son casi universalmente aceptados como LSI.

Todas las direcciones que aparecen sobre la barra de direcciones se generan en la unidad central de procesamiento. Dado que la barra de direcciones tiene 16 bits, pueden generarse direcciones que van desde 0000 0000 0000 0000 (0000_{16}) hasta 1111 1111 1111 1111 ($FFFF_{16}$). Se suele hablar de una "barra de 64 k", que corresponde a 65.536 direcciones diferentes. Cuando el microprocesador genera una dirección, sólo un dispositivo la reconoce y se comunica con el MP. Los 16 bits suelen denominarse A0-A15, designándose como A0 el bit menos significativo, A1 el siguiente, y así sucesivamente hasta A15, el bit más significativo.

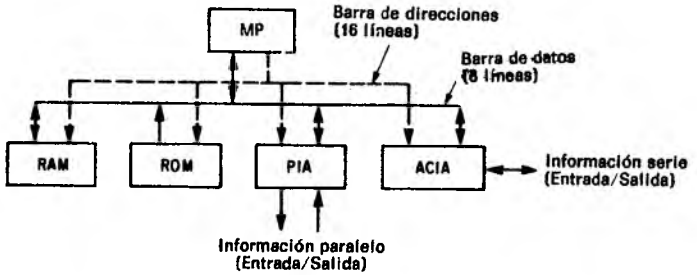


Fig. 9.1 Sistema M6800

La barra de datos de 8 bits es común a todos los dispositivos de la Fig. 9.1. Las flechas de la figura representan el sentido de flujo de la información entre cada dispositivo y el MP. El MP puede recibir información desde la memoria ROM, pero *no puede* enviar información a la misma. En cambio los demás dispositivos del sistema (memorias RAM, interfases para periféricos PIA y ACIA) tienen la capacidad de recibir información del MP y de enviar información hacia el mismo. Las líneas de la barra de datos se conocen como D0-D7, siendo D0 el bit menos significativo, D1 el siguiente, y así sucesivamente hasta llegar a D7, el bit más significativo. En el sistema se conectan entre sí todas las líneas de datos homólogas de todos los dispositivos. Por ejemplo, todas las líneas D0 se conectan entre sí, las líneas D1 entre sí, y así sucesivamente.

Una vez seleccionado un dispositivo, ¿cómo sabe si debe recibir o enviar información al MP? Para ello existe una línea que vincula al MP con todos los dispositivos, conocida como *línea de lectura/escritura* (Read/Write - R/W). Esta línea permite que el MP controle el sentido del flujo de información entre dispositivos. Si la línea R/W toma un nivel lógico alto, el dispositivo debe enviar información al MP; si la línea está baja, el dispositivo debe recibir información desde el MP.

El sistema M6800 puede operar a frecuencias de reloj que van desde 100 kHz hasta 2 MHz, según el modelo, funcionando con una *única* fuente de alimentación de 5 V.

9.2 Unidad de microprocesamiento (MP)

El núcleo de la familia microcomputadora M6300 es la unidad de microprocesamiento M6800. Este elemento viene encapsulado en un circuito integrado de cuarenta terminales, según se aprecia en la Fig. 9.2, y tiene seis registros internos (ver Fig. 9.3) accesibles al usuario:

1. Acumulador A (A)
2. Acumulador B (B)
3. Registro índice (X)
4. Contador de programa (PC)
5. Registro puntero de pila (SP)
6. Registro de códigos de condición (CC)

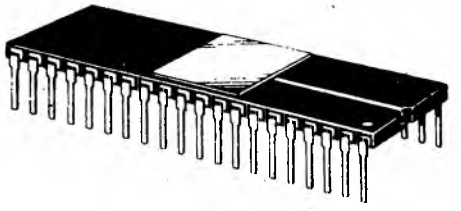


Fig. 9.2 Encapsulado cerámico del MP (40 terminales)

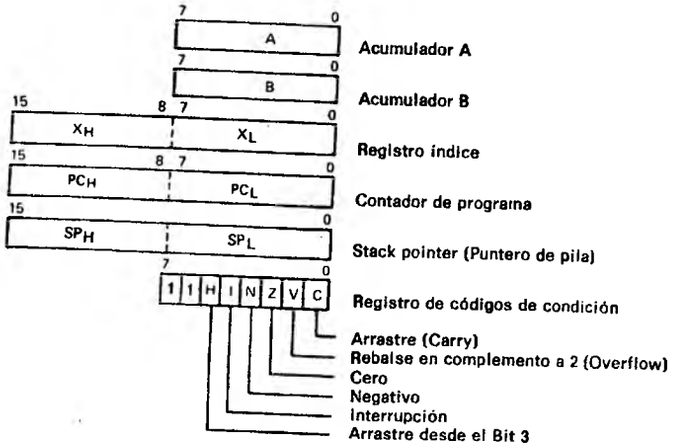


Fig. 9.3 Registros internos del M6800

Registros internos del M6800

1. *Acumulador A (A):* El acumulador A es un registro de 8 bits, que se utiliza como elemento de almacenamiento temporario para las operaciones realizadas por la unidad aritmético-lógica.

2. *Acumulador B (B):* Este registro también es de 8 bits y se utiliza como registro de almacenamiento temporario para operaciones realizadas por la unidad aritmético-lógica.

3. *Registro índice (X):* Este registro es de 16 bits (2 bytes) y se utiliza fundamentalmente para modificar direcciones en el modo de direccionamiento indexado. Puede incrementarse, decrementarse, cargarse desde memoria, almacenarse en memoria o compararse con algún valor dado por programa.

4. *Contador de programa (PC):* Registro de 16 bits que contiene la dirección del próximo byte de la instrucción a leer de memoria. Su valor se incrementa automáticamente cada vez que se trasfiere a la barra de direcciones.

5. *Puntero de pila (SP):* Registro de 16 bits que contiene una dirección de comienzo, normalmente en memoria RAM, para almacenar el estado de los registros del MP cuando éste debe realizar otras funciones, como la atención de una interrupción o salto a subrutina. La dirección almacenada en este registro es la dirección de comienzo de un conjunto de posiciones de memoria ubicadas consecutivamente en la memoria RAM, en las que se almacenan los contenidos de los registros en el siguiente orden:

(Puntero de pila) ← byte inferior de (PC)
(Puntero de pila) – 1 ← byte superior de (PC)
(Puntero de pila) – 2 ← byte inferior de (X)
(Puntero de pila) – 3 ← byte superior de (X)
(Puntero de pila) – 4 ← (A)
(Puntero de pila) – 5 ← (B)
(Puntero de pila) – 6 ← (CC)

Después de almacenar en la pila el contenido de los registros, el puntero de pila se decrementa automáticamente. Cuando se descarga la pila (reponiendo los estados de los registros) la recuperación se realiza en orden inverso al almacenamiento, es decir, se comienza por el último registro almacenado en la pila.

6. *Registro de códigos de condición (CC):* Registro de 8 bits utilizado por las instrucciones de bifurcación, para determinar si el MP debe romper la secuencia de ejecución de las instrucciones. Las bifurcaciones se producen (o no) de acuerdo al estado de determinados bits del registro de códigos de condición. Debe quedar claro que distintas instrucciones afectan los bits de este registro en formas diferentes. Para determinar de qué manera queda afectado cada bit por una instrucción determinada, se sugiere al lector consultar el capítulo 8, donde se ha descrito el juego de instrucciones.

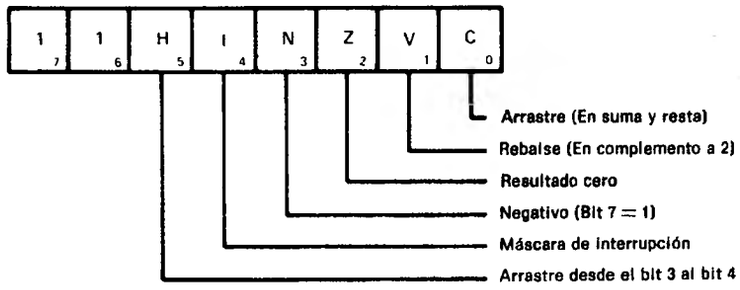


Fig. 9.4 Registro de códigos de condicion

Los bits 6 y 7 de este registro toman siempre el valor "1", y por lo tanto no se utilizan para condicionar bifurcaciones del programa. Los bits restantes, bits 0 a 5, se conocen como H, N, Z, B y C, según se ve en la Fig. 9.4.

6(a). *Bit de arrastre (bit C):* Este bit adopta el valor lógico "1" si al ejecutarse una instrucción se produce un arrastre desde el bit más significativo del resultado de la operación ejecutada. En caso contrario, toma el valor "0".

EJEMPLO UTILIZANDO UNA INSTRUCCION ABA

	ANTES DE LA EJECUCION	DESPUES DE LA EJECUCION
A	1000 0000	0000 0000
B	1000 0000	1000 0000
CC (C)	0 0 1	1

6(b). *Bit de desborde (bit V):* Este bit del registro de códigos de condición adopta el valor "1" si como resultado de una operación aritmética se produce un desborde de la capacidad del acumulador en complemento a 2; adopta el valor "0" en caso contrario. Se produce desborde en complemento a 2 cuando el resultado de una operación excede el rango ± 127 en un registro de ocho bits.

EJEMPLO UTILIZANDO UNA INSTRUCCION DEC A

	ANTES DE LA EJECUCION	DESPUES DE LA EJECUCION
A	1000 0000	0111 1111
CC (V)	0 0 1	1

Debe recordarse que la operación de resta se resuelve mediante la suma del complemento a 2. Por lo tanto, para decrementar en "1" un registro (instrucción DEC), debe sumarse al contenido del registro a decrementar el complemento a 2 del número 1:

Número	→	0000 0001
Complemento a 1	→	1111 1110
Complemento a 2	=	1111 1111
Contenido del registro A	= +	1000 0000
Complemento a 2 del resultado	=	1 0111 1111

Dado que hay desborde, el bit V toma el valor "1".

6(c). *Bit del cero (bit Z):* Este bit del registro de códigos de condición toma valor "1" si el resultado de una operación aritmética es "0"; en caso contrario adopta el valor "0".

EJEMPLO UTILIZANDO UNA INSTRUCCION DEC A

	ANTES DE LA EJECUCION	DESPUES DE LA EJECUCION
A	0000 0001	0000 0000
CC (Z)	0 0 1	1
O...		
A	0000 0010	0000 0001
CC (Z)	0 0 1	0

6(d). *Bit de negativo (bit N):* Este bit adopta el valor "1" si el resultado de una operación da un número negativo (bit más significativo del resultado en "1"), y "0" en caso contrario

EJEMPLO UTILIZANDO UNA INSTRUCCION LDA A #$80

	ANTES DE LA EJECUCION	DESPUES DE LA EJECUCION
A	0000 0000	1000 0000
CC (N)	0 0 1	1

6(e). *Máscara de interrupciones (bit I):* Cuando este bit adopta el valor lógico "1" se inhiben todas las interrupciones enmascarables ($\overline{IRQ}$) que llegan al MP, en cambio si adopta el nivel "0" el MP admite interrupciones cuando la línea $\overline{IRQ}$ adopta el nivel lógico bajo. El bit I puede llevarse a "1" con la instrucción SEI; será llevado automáticamente a "1" por el MP cuando se produzca una interrupción, o cuando se decodifique una instrucción SWI. Este bit toma el valor "0" cuando el MP ejecuta una instrucción RTI, si la interrupción había sido causada por la línea $\overline{IRQ}$, o bien mediante la instrucción CLI.

6(f). *Bit de arrastre intermedio (bit H):* Este bit adopta el valor "1" si, durante una operación aritmética correspondiente a las instrucciones ABA, ADC o ADD, hay arrastre desde el bit 3 hacia el bit 4 del resultado. En caso contrario el bit H toma el valor "0"

EJEMPLO UTILIZANDO UNA INSTRUCCION ABA

	ANTES DE LA EJECUCION	DESPUES DE LA EJECUCION
A	0000 1000	0001 0000
B	0000 1000	0000 1000
CC (H)	0 6 1	1

6(g). *Bit 6 y bit 7:* Siempre igual a "1".

Señales de entrada/salida del M6800

Como ya se ha mencionado, el microprocesador M6800 es un dispositivo de cuarenta terminales; en la Fig. 9.5 se ilustra la distribución de terminales, indicándose mediante flechas el sentido de flujo de la información en cada línea. Nótese que no se utilizan los terminales 35 y 38.

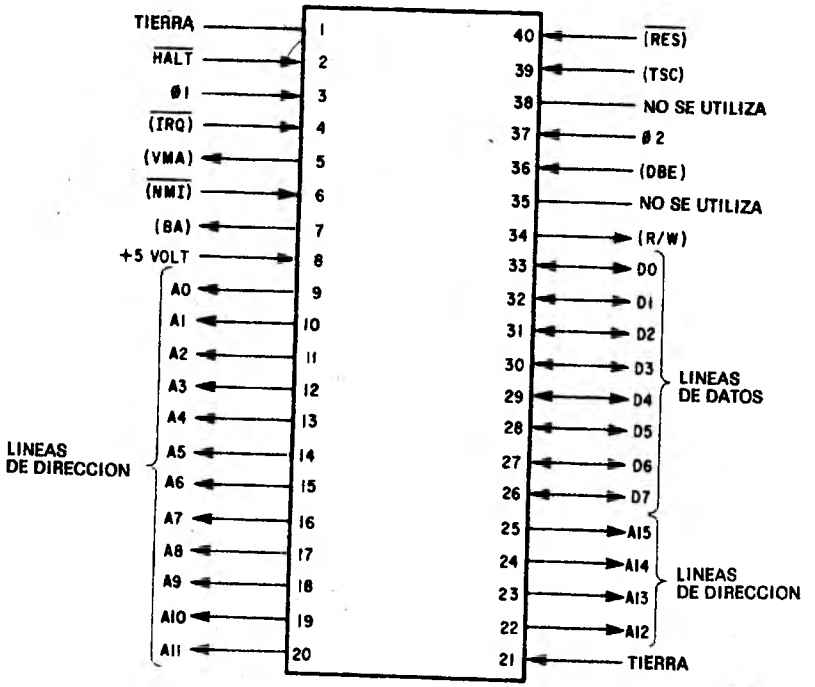


Fig. 9.5 Señales del M6800

1. *Masa (terminales 1 y 21):* Estos terminales deben conectarse a la masa del sistema, que corresponde al terminal negativo de la fuente de alimentación de 5 V.

2. *HALT (terminal 2):* Cuando esta línea tiene un nivel lógico 1" el MP funciona normalmente. Cuando la línea HALT adopta un nivel lógico "0", el MP completa la instrucción que está ejecutando antes de detenerse. Al detenerse, la línea "barra disponible" (Bus Available - BA) toma nivel "1", la línea "dirección válida" (Valid Memory Address - VMA) pasa a "0", y la línea "lectura/escritura" (Read-Write - RW), así como las barras de dirección y datos adoptan un estado de alta impedancia. Si se produce una instrucción (IRQ o NMI) mientras el MP está detenido, queda almacenada en el MP hasta que el mismo sale del estado de detención, momento en el cual es atendida. La línea HALT permite que una fuente externa controle la ejecución de un programa ejecutando una instrucción por vez. Esta prestación es particularmente útil durante la etapa de depuración y prueba de un programa. Durante la operación normal del sistema, este terminal debe conectarse a +5 V.

3. *Fase 1 (terminal 3):* Esta línea es la entrada de la fase 1 del reloj del sistema.

4. *Línea de interrupción (IRQ) (terminal 4):* Cuando algún dispositivo externo baja el nivel lógico de la línea IRQ, el MP, tras completar la instrucción que está ejecutando, va a ejecutar una secuencia de atención de interrupciones (siempre y cuando no se haya inhibido la atención de interrupciones colocando en "1" la máscara correspondiente). Como primer paso de esta secuencia se almacenan en memoria RAM los contenidos del registro índice, contador de programa, los dos acumuladores y el registro de códigos de condición, para poder recuperarlos después de haber sido atendida la interrupción. Luego se coloca en "1" el bit I del registro de códigos de condición, para inhibir posteriores interrupciones. El MP carga en el contador de programa el contenido de las direcciones de memoria $FFF8_{16}$ y $FFF9_{16}$ (corresponden a la dirección más alta de memoria ROM −6 y −7); el contenido de estas dos posiciones de memoria es la dirección del programa que atiende la interrupción y determina las acciones que debe tomar el MP ante la interrupción. En algún punto de la subrutina se encuentra una instrucción RTI, que hace que el microprocesador y sus registros internos se repongan a la condición en que estaban antes de recibirse el pedido de interrupción. La Fig. 9.6 presenta un diagrama de flujo detallado de la secuencia de interrupción.

La secuencia correspondiente a una interrupción enmascarable (IRQ) es la siguiente:

1) Se ingresa en la secuencia de atención si el bit I del registro de códigos de condición no está en "1", y si la línea $\overline{IRQ}$ toma un valor lógico "0" durante al menos un ciclo de reloj de fase 2.
2) Al completarse la instrucción en ejecución, se almacenan en memoria RAM los registros internos (contador de programa, índice, acumulador A, acumulador B y registro de códigos de condición) a partir de la dirección indicada por el puntero de pila, y en sentido decreciente de direcciones de memoria (siete bytes en total).
3) Se coloca en "1" la máscara de interrupciones (bit I del registro de códigos de condición).
4) Se carga en la mitad superior del contador de programa (PCH) el contenido de la dirección $FFF8_{16}$.
5) Se carga en la mitad inferior del contador de programa (PCL) el contenido de la dirección $FFF9_{16}$.
6) Durante la fase 1 de reloj se coloca el nuevo contenido del contador de programa en la barra de direcciones.
7) Durante la fase 2 del reloj el contenido de la dirección indicada se carga en el registro de instrucción y se decodifica como primera instrucción de la rutina de interrupción.
8) Si la instrucción decodificada toma más de un byte, se leen los bytes restantes de la instrucción para su ejecución.
9) Luego de la ejecución de la instrucción se repite el paso 7, hasta completar todas las instrucciones de la rutina de interrupción (lo cual se detecta a través de la instrucción RTI).

La secuencia correspondiente a una instrucción SWI es la siguiente:

1) Se almacenan los registros del MP en memoria RAM a partir de la dirección indicada por el puntero de pila (siete bytes en total).
2) Se coloca en "1" la máscara de inhibición de interrupciones.
3) Se carga la mitad superior del contador de programa (PCH) con el contenido de la dirección $FFFA_{16}$.
4) Se carga en la mitad inferior del contador de programa (PCL) el contenido de la dirección $FFFB_{16}$.
5) Se envía el contenido del contador de programa en la barra de direcciones durante la fase 1 del reloj.
6) Durante la fase 2 del ciclo del reloj se decodifica el contenido de la dirección anterior como primera instrucción de la subrutina de atención de SWI.

7) Si la instrucción tiene más de un byte se leen los bytes siguientes para su ejecución.
8) Después de la ejecución se repite el paso 6 para ejecutar toda la rutina de interrupción, la que se completa al detectarse una instrucción RTI.

5. *Dirección de memoria válida (VMA) (terminal 5):* Esta señal en estado alto informa a todos los dispositivos conectados a la barra de direcciones que en el mismo existe una dirección válida. Esta señal sólo puede adoptar los dos valores lógicos de "0" y de "1", no pudiendo tomar un estado de alta impedancia.

6. *Interrupción no enmascarable ($\overline{NMI}$) (terminal 6):* Es similar a la entrada $\overline{IRQ}$, pero no puede enmascararse. Al igual que en las interrupciones por $\overline{IRQ}$, el MP completa la instrucción que está ejecutando antes de atender la interrupción. Luego de guardar en memoria los contenidos de los registros para su posterior uso, se pone en "1" el bit I del registro de códigos de condición y se carga en el contador de programa el contenido de las direcciones de ROM, $FFFC_{16}$ y $FFFD_{16}$. Estas posiciones contienen la dirección de comienzo del programa de atención de la interrupción no enmascarable. Cuando el MP encuentra una instrucción RTI, recupera el contenido de sus registros para reponer la condición en la que se hallaba antes de la llegada del pedido de interrupción. La Fig. 9.6 presenta un diagrama de flujo detallado de la secuencia de interrupción, que es la siguiente:

1) Si la línea $\overline{NMI}$ baja por lo menos durante un ciclo de fase 2 del reloj, el MP completa la instrucción que está ejecutando.
2) Se almacenan en memoria RAM los registros internos (contador de programa, índice, acumuladores A y B, y registro de códigos de condición), a partir de la dirección indicada por el puntero de pila y en sentido de direcciones decrecientes (siete bytes en total).
3) Se coloca en "1" la máscara de interrupciones.
4) Se carga en el byte superior del contador de programa el contenido de la dirección $FFFC_{16}$.
5) Se carga en el byte inferior del contador de programa el contenido de la dirección $FFFD_{16}$.
6) Durante la fase 1 de reloj se coloca el contenido del contador de programa sobre la barra de direcciones.
7) Durante la fase 2 del reloj el contenido de la dirección seleccionada se carga en el registro de instrucción, decodificándose como primera instrucción de la subrutina de atención de $\overline{NMI}$.

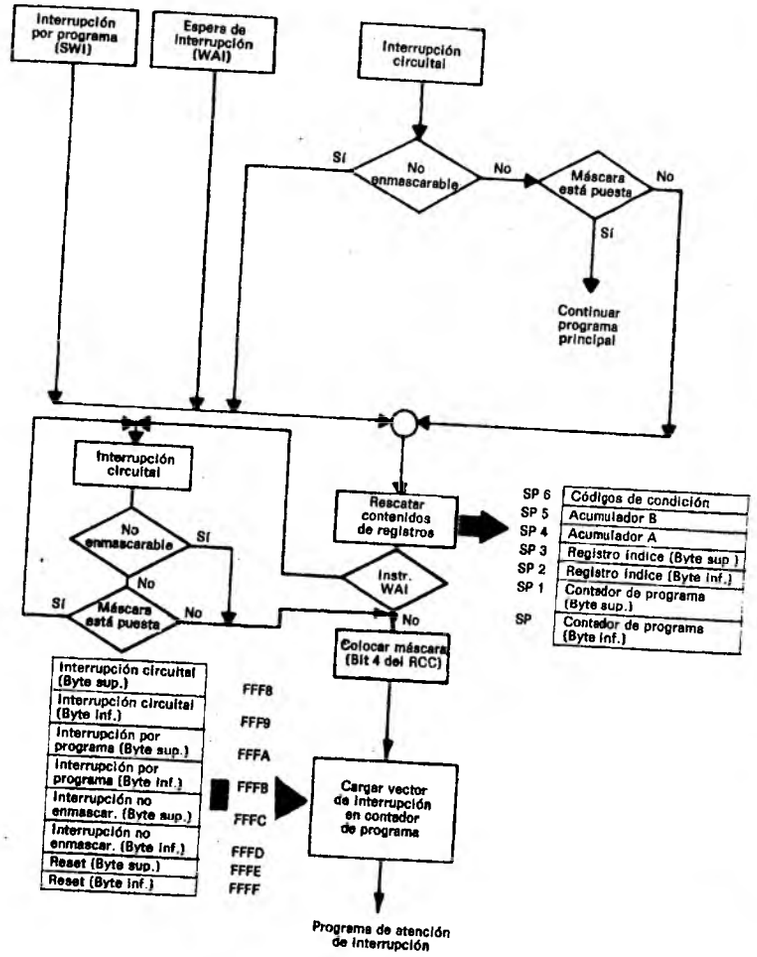


Fig. 9.6 Diagrama de flujo de interrupciones

8) Si la instrucción lleva más de un byte, se cargan los bytes adicionales antes de ejecutarla.

9) Luego de ejecutar la instrucción se repite el paso 7 hasta ejecutar la totalidad de la rutina, lo que se detecta al decodificar la instrucción RTI.

7. *Barras disponibles (BA) (terminal 7):* En funcionamiento normal, esta línea adopta un nivel lógico "0" indicando que las líneas de datos y direcciones están bajo control del MP. Cuando esta línea pasa a "1" las líneas de dirección y datos se encuentran disponibles

para su uso por parte de un dispositivo exterior, lo que significa que el MP las deja en alta impedancia. Esta condición se presenta cuando el MP está detenido (línea $\overline{\text{HALT}}$ en nivel bajo) o si, como resultado de una instrucción WAI, está en situación de espera.

8. *Alimentación 5 V (terminal 8):* Corresponde al terminal positivo de la fuente de alimentación del sistema.

9. *Líneas de dirección (A0-A15) (terminales 9 a 20 y 22 a 25):* Estas dieciséis líneas de salida se utilizan para selección de dispositivos externos al microprocesador. Las direcciones se generan en la unidad de proceso. Las salidas de las dieciséis líneas son excitadores de barra con capacidad de adoptar un estado de alta impedancia y de manejar una carga TTL y 130 pF a 1 MHz. Cuando la salida se inhabilita, el circuito queda abierto.

10. *Líneas de datos (D0-D7) (terminales 26 a 33):* Estas ocho líneas bidireccionales se usan para trasferir información entre el MP y los dispositivos periféricos. Pueden colocarse en estado de alta impedancia y tienen elementos de salida capaces de manejar una carga TTL común y 130 pF a 1 MHz.

11. *Lectura/Escritura (R/W) (terminal 34):* Indica a todos los dispositivos externos que el MP se encuentra en estado de lectura, si la línea está en "1", o de escritura si la línea está en "0". El estado normal de esta línea corresponde al nivel "1" (lectura), y toma un estado de alta impedancia bajo control de la señal correspondiente (TSC, terminal 39) o en el caso de detención del MP.

12. *Habilitación de la barra de datos (DBE) (terminal 36):* Habilita los excitadores de salida de la barra de datos cuando su estado lógico es "1". Normalmente se genera esta entrada a partir de la señal de fase 2 del reloj; en su estado alto permite la salida de información hacia la barra de datos durante los ciclos de escritura del MP. Durante los ciclos de lectura, los excitadores se inhiben internamente.

13. *Fase 2 del reloj (terminal 37):* Se inyecta en esta entrada la señal de fase 2 del reloj del sistema, Ø 2.

14. *Control de alta impedancia (TSC) (terminal 39):* Cuando esta línea está en "1" fuerza las líneas de dirección y la línea R/W al estado de alta impedancia. Las líneas VMA y BA pasan a "0", evitando falsas lecturas o escrituras en dispositivos controlados por la línea VMA. La barra de datos no sufre la acción de esta señal, dado que tiene su propio sistema de habilitación. Cuando se pone en "1" la línea TSC, debe mantenerse la línea fase 1 en "1" y la línea fase 2 en "0", para detener la ejecución del programa y liberar la barra de direcciones para su uso por otros dispositivos. Dado que el MP es

un dispositivo dinámico, debe tenerse la precaución de no mantenerlo en condición de alta impedancia por más de 9,5 μseg, bajo riesgo de destruir la información almacenada en el mismo.

15. *Reset ($\overline{RES}$) (terminal 40):* Se utiliza para arrancar el MP cuando se conecta la alimentación al sistema. Luego de activarse la alimentación y de que ésta alcance 4,75 V como mínimo, la línea $\overline{RES}$ debe mantenerse en "0" durante al menos ocho ciclos de reloj, tiempo durante el cual la barra de direcciones contiene la dirección $FFFE_{16}$.

Tabla 9.1 Resumen de señales del M6800

TERMINAL Nº	DESCRIPCION DE LA SEÑAL	NOMBRE DE LA SEÑAL	TIPO DE SEÑAL	ALTA IMPEDANCIA
1	Tierra	GND	Entrada	No
2	Detención del MP	$\overline{HALT}$	Entrada	No
3	Fase 1 del reloj	Ø1	Entrada	No
4	Interrupción	$\overline{IRQ}$	Entrada	No
5	Dirección válida de memoria	VMA	Salida	No
6	Interrupción no enmascarable	$\overline{NMI}$	Entrada	No
7	Barra de datos disponible	BA	Salida	No
8	Alimentación	+ 5	Entrada	No
9-20, 22-25	Líneas de dirección	A0-A15	Salida	Sí
21	Tierra	GND	Entrada	No
26-33	Líneas de datos	D0-D7	Entrada/Salida	Sí
34	Lectura/Escritura	R/W	Salida	Sí
36	Habilitación de la barra de datos	DBE	Entrada	No
37	Fase 2 del reloj	Ø2	Entrada	No
39	Control de alta impedancia	TSC	Entrada	No
40	Inicialización	$\overline{RES}$	Entrada	No

Luego de un mínimo de ocho ciclos de reloj, puede permitirse que la línea $\overline{RES}$ suba a "1", indicando así al MP que inicie la secuencia de encendido (restart). El contador de programa se cargará en su byte más significativo con el contenido de la dirección $FFFE_{16}$, y en su byte menos significativo por el contenido de la dirección $FFFF_{16}$. En este momento el contador de programa contiene la dirección de comienzo de la rutina de inicialización. Durante la secuencia de inicialización (Fig. 9.7) la máscara de interrupciones (bit I del registro de códigos de condición) toma el valor "1". La habilitación de las interrupciones puede insertarse en el procedimiento de inicialización mediante la instrucción CLI.

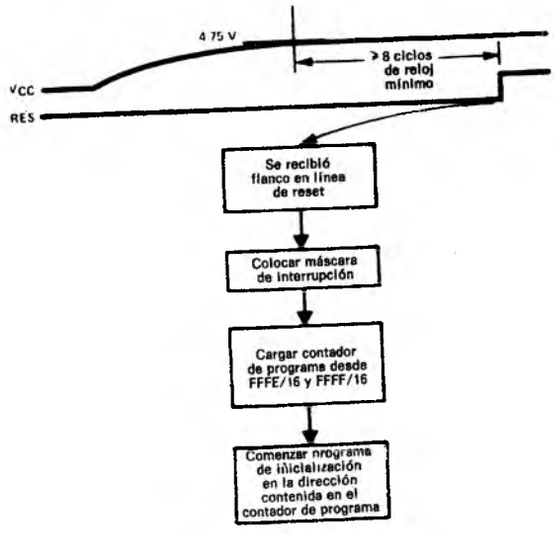


Fig. 9.7 Secuencia de arranque (Reset)

La secuencia inicial es la siguiente:

1. Con la señal $\overline{HALT}$ en "1", la línea $\overline{RES}$ baja durante al menos ocho ciclos de fase 1 y fase 2, tiempo durante el cual el bit I se coloca en "1".
2. El contador de programa se carga en su byte superior con el contenido de la dirección $FFFE_{16}$.
3. El contador de programa se carga en su byte inferior con el contenido de la dirección $FFFF_{16}$.
4. Durante la fase 1 se coloca el contenido del contador de programa sobre la barra de direcciones.

Reloj del sistema

Según se ha mencionado en el capítulo 5, todos los sistemas digitales deben tener una señal de reloj que sincronice la ejecución de las funciones a realizar. El sistema microcomputador M6800 no es una excepción en este sentido. Requiere un reloj bifásico, de fases no superpuestas (Ø 1 y Ø 2 no pueden valer "1", simultáneamente), capaz de operar a partir de la fuente de alimentación de 5 V. La Fig. 9.8 muestra las formas de ondas y especificaciones.

Esta sección describe la operación del reloj del sistema, y muestra cómo el mismo controla todas las operaciones del MP. Dentro del MP existe un registro, llamado *registro de instrucción (IR)*, que sirve