MOTOROLA

Iniciación a los Microcontroladores de las familias de 8 bits

Jordi Mayné

Ingeniero de Aplicaciones

SILICA®
An Avnet Division

Rev. 2

Índice

¿Qué es un Microcontrolador?

Índice

Introducción

Este capítulo pone el fundamento para una exploración detallada del funcionamiento interno de un pequeño microcontrolador. Se podrá ver que el microcontrolador es una de las formas más básicas de un sistema con microprocesador. Aunque son mucho más pequeños que los microprocesadores personales y los grandes ordenadores, se construyen microcontroladores con los mismos elementos básicos. En el sentido más simple, los microprocesadores producen un modelo específico basado en unas entradas y unas salidas, con las instrucciones en un programa con microprocesador.

Como la mayoría de microprocesadores, los microcontroladores son simplemente ejecutores de instrucciones de propósito general. La estrella real de un sistema con microprocesador, es un programa de instrucciones que son proporcionadas por un programador. Este programa le dice al microprocesador que realice largas secuencias de acciones muy simples para lograr tareas útiles como las pensadas por el programador.

Vista global de un Sistema con Microprocesador

La Figura 1 proporciona una vista global de un sistema con microprocesador. Simplemente cambiando los tipos de dispositivos de entrada y de salida, éste diagrama de bloques podría ser el de un **microprocesador personal**, un **ordenador** o un simple un **microcontrolador** (MCU). Los dispositivos de entrada y de salida (**E/S**) mostrados en la figura son lo típicos encontrados en un sistema con microprocesador.

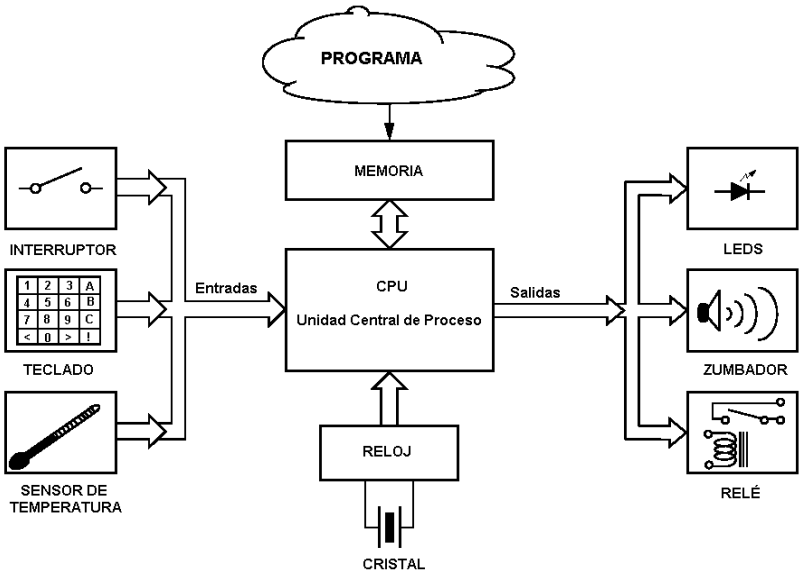


Figura 1. Sistema típico con Microprocesador

Entradas de un Sistema con Microprocesador

Los dispositivos de entrada proporcionan información del mundo exterior al sistema con microprocesador. En un ordenador personal, el dispositivo de entrada más común es el teclado, igual que una máquina de escribir. Los sistemas con microprocesadores normalmente usan dispositivos de entrada mucho más simples como interruptores o pequeños teclados, aunque los dispositivos de entrada más exóticos se encuentran en los sistemas basados en microprocesador. Un ejemplo de un dispositivo de entrada exótico para un microprocesador es el sensor de oxígeno en un automóvil que mide la eficacia de la combustión tomando muestras en el tubo de escape.

La mayoría de entradas del microprocesador pueden procesar sólo señales de entrada **digital,** al mismo nivel de voltaje que el de la fuente de alimentación. El nivel 0 V o tierra se le llama $\mathbf{V_{SS}}$ y el nivel positivo de la fuente de alimentación se le llama $\mathbf{V_{DD}}$ y es típicamente de 5 Vdc. Un nivel aproximado de 0 voltios indica una señal **lógica '0'** y un voltaje aproximadamente igual al nivel positivo de la fuente de alimentación indica una señal **lógica '1'**.

Por supuesto, el mundo real está lleno de señales analógicas o señales que son de otros niveles de voltaje. Algunos dispositivos de entrada traducen los voltajes de señal de algún otro nivel a los niveles $\mathbf{V_{DD}}$ y $\mathbf{V_{SS}}$, necesarios para el microprocesador. Otros dispositivos de entrada convierten las señales analógicas en señales digitales (a valores binarios '1' y '0') para que el microprocesador los pueda entender y manipular. Algunos microprocesadores incluyen circuitos convertidores analógicos/digitales en el mismo circuito integrado.

Los **transductores** se pueden usar para traducir otras señales del mundo real a niveles de señal lógica que un microprocesador puede entender y manipular. Algunos ejemplos que incluyen transductores, como los sensores de temperatura, sensores de presión, detectores de nivel de luz y otros. Con estos transductores, casi cualquier propiedad física se puede usar como entrada a un sistema con microprocesador.

Salidas de un Sistema con Microprocesador

Se usan dispositivos de salida para comunicar la información o acciones del sistema con microprocesador al mundo exterior. En un ordenador personal, el dispositivo de salida más común es la pantalla CRT(tubo de rayos catódicos). Los sistemas con microprocesador usan a menudo dispositivos de salida mucho más simples como los LEDs, lámparas, o zumbadores. Circuitos convertidores (a veces construidos en el mismo circuito integrado microprocesador) pueden convertir señales digitales a niveles de voltaje analógicos.

Del "controlador" en microcontrolador viene del hecho de que estos pequeños sistemas con microprocesador normalmente controlan algo en comparación con un ordenador personal que normalmente procesa información. En el caso del ordenador personal, la mayoría de las salidas es de información (cualquier información en una pantalla CRT o en el papel de la impresora). Por otro lado, en un sistema con microprocesador, la mayoría de las salidas son señales de nivel lógico digital, que se usan para manejar LEDs o dispositivos eléctricos como relés o motores.

Unidad Central de Proceso (CPU)

La CPU (Central Processor Unit) es el centro de cada sistema microprocesador. El trabajo de la CPU es ejecutar obedientemente las instrucciones de un programa que le fue proporcionado por un programador. Un **programa** con microprocesador le dice a la CPU que lea (read) la información de las entradas y que la escriba (write) a la memoria de trabajo o que lea la información de la memoria de trabajo y la escriba a las salidas. Algunas instrucciones del programa involucran decisiones simples que causan al programa continuar con la siguiente instrucción o saltar a un nuevo lugar del programa. En un capítulo posterior, se verán de cerca el juego de instrucciones disponibles para un microcontrolador en particular.

En un ordenador personal, hay varios niveles de programas, empezando con el programa interno, que es el control más básico del funcionamiento del microprocesador. Otro nivel incluye programas de usuario que se cargan en la memoria del sistema cuando están a punto de ser usados. Esta estructura es muy compleja y no sería un buen ejemplo para mostrar a un principiante cómo trabaja el microprocesador.

En un microcontrolador normalmente, solamente un programa en particular está trabajando para el control de una aplicación. Por ejemplo, la CPU MC68HC05 sólo reconoce 60 **instrucciones** diferentes, pero éstas son representativas del juego de instrucciones de cualquier sistema con microprocesador. Este tipo de sistema con microprocesador es un buen modelo para aprender el fundamento de funcionamiento de un microprocesador, porque es posible saber lo que está pasando exactamente en cada paso de la ejecución de un programa en la CPU.

Reloj (Clock)

Salvo excepciones, los microprocesadores usan un pequeño **oscilador** del reloj (clock) para activar la CPU, para mover de un paso a la secuencia siguiente. En el capítulo de arquitectura de un microprocesador, se puede ver que incluso las instrucciones simples de un microcontrolador están compuestas de una serie de pasos aún más básicos. Cada uno de estos pasos diminutos en el funcionamiento del microprocesador toma un ciclo del reloj de la CPU.

Memoria del Microprocesador

Se usan varios tipos de memoria para los diferentes propósitos en un sistema con microprocesador. Los tipos principales de memoria encontrados en un microcontrolador son:

Memorias para almacenar el Programa:

- **ROM** (Read Only Memory): memoria sólo de lectura, este tipo de memoria se programa en fabrica y se llama **Máscara**.
- **EPROM** (Erasable Programmable Read Only Memory): memoria sólo de lectura, programable eléctricamente y se borra por luz ultravioleta a través de una ventana en la parte superior del dispositivo.
- **OTP** (One Time Programmable): memoria sólo de lectura, programable eléctricamente una sola vez.
- **FLASH**: memoria programable y borrable eléctricamente, por bloques.

Estos tipos se usan principalmente para almacenar los programas y los datos permanentes que deben permanecer inalterados incluso cuando no hay ninguna alimentación aplicada al microcontrolador.

Memoria para almacenar Datos:

- **RAM** (Random Access read/write Memory): memoria de acceso a lectura o escritura aleatorio, se usa para el almacenamiento temporal de datos y el cálculo intermedio de los resultados durante las operaciones. Este tipo de memoria pierde los datos cuando se queda sin alimentación.
- **EEPROM** (Electrically Erasable Programmable Read Only Memory): memoria sólo de lectura, programable y borrable eléctricamente.

La unidad más pequeña de una memoria, es de un solo **bit,** que puede guardar uno valor lógico ‘0’ o ‘1’. Estos bits se agrupan en conjuntos de ocho bits para hacer uno **byte**. Los microprocesadores más grandes utilizan grupos de 16 o 32 bits, llamados ‘palabras’ o **word**. El tamaño de una ‘palabra’ (word) puede ser diferente para cada microprocesador, pero un byte siempre es de ocho bits.

Los ordenadores personales trabajan con programas muy grandes y con grandes cantidades de datos, para ello usan formas especiales de dispositivos de almacenamiento, llamados almacenamiento en masa, como los discos blandos, los discos duros, y los discos compactos. No es raro encontrar varios millones de bytes de memoria RAM en un ordenador personal, con discos duros con varios gigabytes o discos compactos muy similares a los usados para las grabaciones de música con una capacidad de 640 millones de bytes de memoria de sólo lectura. En comparación, los sistemas con microcontrolador típico tienen una memoria total entre 1,000 y 64,000 bytes.

Programa de un Microprocesador

La Figura 1 muestra el programa como una nube, porque se origina en la imaginación del ingeniero o programador del microprocesador. Esto es comparable a un ingeniero eléctrico que piensa en un nuevo circuito o un ingeniero mecánico que deduce un nuevo ensamblaje. Los componentes de un programa son las instrucciones del juego de instrucciones de la CPU. Así como el diseñador del circuito puede construir un circuito sumador con simple puertas AND, OR y NOT, un programador puede escribir un programa para sumar números con simples instrucciones.

Los programas se guardan en la memoria de un microprocesador donde pueden ser ejecutados de modo secuencial por la CPU. En el capítulo de programación, se aprenderá a escribir programas y prepararlos para ser cargados en la memoria de un microprocesador.

El Microcontrolador

Hasta ahora se han visto varias partes de un sistema con microprocesador y ya se está preparado para hablar sobre los microcontroladores. En la mitad superior de la Figura 2 se muestra un sistema con microprocesador genérico, con una parte adjunta de contorno punteado. Esta parte, es un microcontrolador y la mitad inferior de la figura es un diagrama de bloques que muestra su estructura interior con más detalle. El cristal no se contiene dentro del microcontrolador, pero es una parte necesaria del circuito oscilador. En algunos casos, se puede substituir el cristal por un resonador cerámico que es más económico o un aún menos caro con un conjunto RC (resistencia-condensador).

Un **microcontrolador** puede definirse como un sistema microprocesador completo, que incluye la CPU, la memoria, un oscilador del reloj, las E/S y otros periféricos en un solo circuito integrado. Cuando algunos de estos elementos como las E/S o la memoria no están incluidos, al circuito integrado se le llama **microprocesador**. La CPU de un ordenador personal es un microprocesador.

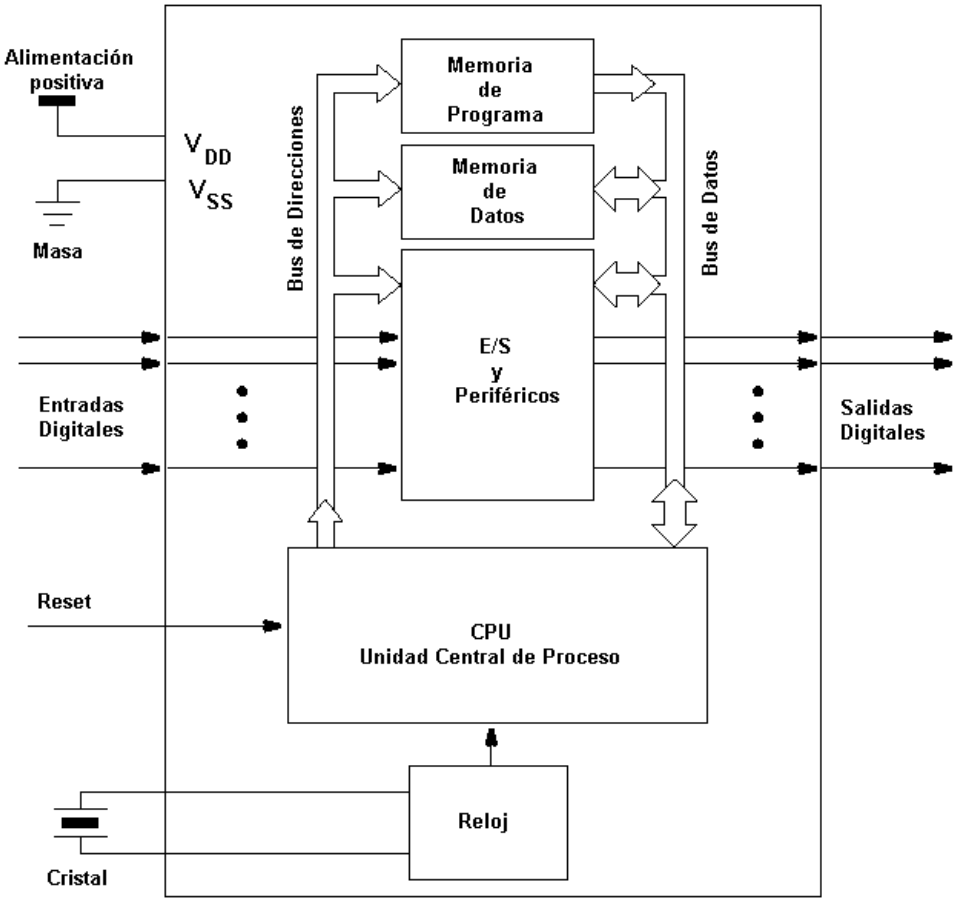


Figura 2. Vista extendida de un Microcontrolador

Resumen

Un **microcontrolador** es un sistema microprocesador completo, incluyendo la CPU, la memoria, el oscilador del reloj y las E/S en un solo circuito integrado.

Partes de cualquier microprocesador

- Una unidad central de proceso o **CPU**
- Un **Reloj** de secuencia de la CPU
- **Memorias** para las instrucciones y los datos
- **Entradas** para poner la información al sistema
- **Salidas** para sacar la información del sistema
- **Programa** para hacer algo útil con el microprocesador

Tipos de Microprocesadores

Aunque todos los microprocesadores comparten los mismos elementos básicos e ideas, hay diferentes tipos de microprocesadores para diferentes propósitos.

- Para los grandes ordenadores se usan microprocesadores muy grandes, que se usan para procesar grandes informaciones.
- Para los ordenadores personales se usan microprocesadores cada vez más grandes, para hacer desde las tareas más pequeñas como el procesador de texto, hasta el diseño asistido.
- Para el control de muchos aparatos se usan los microcontroladores de un sólo circuito, que pueden ser desde muy pequeños en prestaciones (por ejemplo, el ratón de un ordenador personal) hasta controles industriales bastante complejos, debido a la alta integración de los mismos.

Números y Códigos de un microprocesador

Índice

Introducción

Este capítulo discute los números binarios, hexadecimales, octales y BCD que normalmente son usados por los microprocesadores.

Los microprocesadores trabajan mejor con la información en una forma diferente que lo usan las personas, para resolver problemas. Los humanos típicamente trabajamos con el sistema de numeración en base 10 (decimal), probablemente porque tenemos 10 dedos. Los microprocesadores trabajan con el sistema de numeración en base 2 (binario), porque esto permite representar toda la información por dos dígitos, que sólo pueden ser '0' o '1'. A su vez, un **1** o un **0** puede ser representado por la presencia o la ausencia de voltaje lógico en una línea de señal o en los estados 'cerrado' (on) o 'abierto' (off) de un simple interruptor.

Los microprocesadores también usan códigos especiales para representar la información alfabética y las instrucciones del microprocesador. Entendiendo estos códigos ayudará a entender cómo los ordenadores pueden hacer tantas cosas, con las cadenas de dígitos que sólo pueden ser '1' o '0'.

Números Binarios y Hexadecimales

Los números **decimales** (**base 10**), el peso de cada dígito es 10 veces más grande que el dígito inmediato, el de su derecha. El dígito de más a la derecha de un número entero decimal es el de la unidad, el dígito de la izquierda es el dígito de las decenas, y así sucesivamente.

Los números **binarios** (**base 2**), el peso de cada dígito es 2 veces más grande que el dígito inmediato de su derecha. El dígito de más a la derecha de un número entero binario es el de la unidad, el siguiente dígito a la izquierda es el dígito cuatro, el siguiente dígito es el ocho y así sucesivamente.

Aunque los microprocesadores funcionan bastante cómodos con los números binarios de 8, 16, o incluso 32 dígitos binarios, los humanos lo encuentran inoportuno trabajar con tantos dígitos en cada momento. El sistema de numeración en **base 16** (**hexadecimal**) ofrece un compromiso práctico. Un dígito hexadecimal se puede representar exactamente con cuatro dígitos binarios, así que un número binario de 8 bits puede ser expresado por dos dígitos en hexadecimal.

La correspondencia entre un dígito hexadecimal y los cuatro dígitos binarios que representan, son bastante simples para los humanos. Los que trabajan con microprocesadores aprenden a traducir mentalmente entre los dos sistemas muy fácilmente. Los números en hexadecimal (base 16), el peso de cada dígito es 16 veces más grande que el dígito inmediato de su derecha. El dígito de más a la derecha de un número entero hexadecimal es el de la unidad, el siguiente dígito a la izquierda es el dígito 16, y así sucesivamente.

La Tabla 1 muestra la relación entre los valores representados en decimal, binario y hexadecimal. Estos tres diferentes sistemas de numeración son simplemente maneras diferentes de representar las mismas cantidades físicas. Las letras desde la A hasta la F se usan para representar los valores hexadecimales correspondientes del 10 hasta el 15, porque cada uno de los dígitos hexadecimales puede representar 16 cantidades diferentes; considerando que, los números de normalmente utilizados sólo incluyen 10 únicos símbolos (del 0 al 9), se tuvieron que utilizar otros símbolos de un solo dígito para representar los valores hexadecimales, del 10 hasta 15, con las letras (A - F).

Decimal (base 10)	Binario (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
..	..	..
100	0110 0100	64
101	0110 0101	65
..	..	..
255	1111 1111	FF
1024	0100 0000 0000	400
1025	0100 0000 0001	401
..	..	..
65.535	1111 1111 1111 1111	FFFF

Tabla 1. Equivalencias entre Decimal, Binario y Hexadecimal

Para evitar cualquier confusión sobre si un número es hexadecimal o decimal, se pone el símbolo '$' antes del número hexadecimal. Por ejemplo, 64 en decimal significa "sesenta y cuatro" y $64 en hexadecimal significa "seis-cuatro", qué es equivalente a 100 en decimal. Algunos fabricantes de microprocesadores siguen a los valores hexadecimales con una H (tal como 64H).

El sistema de numeración Hexadecimal es una buena manera para expresar y discutir el proceso de la información numérica para los microprocesadores, porque mentalmente es fácil para las personas convertir entre los dígitos hexadecimales y los 4-bits binarios equivalentes. La anotación hexadecimal es mucho más compacta que en binario.

Códigos ASCII

Los microprocesadores deben manejar muchos tipos de información diferente de los números. El texto (los caracteres alfanuméricos) y las instrucciones se deben poner en un código, de semejante manera, para que el microprocesador pueda entender esta información. El código más común para la información de tipo texto, es el código estándar para el intercambio de información americano (o **ASCII**).

El código ASCII establece una correlación ampliamente aceptada entre los caracteres alfanuméricos y los valores binarios específicos. Usando el código ASCII $41, corresponde a la letra mayúscula A, $20 corresponde a un carácter de espacio, etc. El código ASCII traduce caracteres a códigos binarios de 7-bits, pero en la práctica la información de los caracteres se lleva normalmente con 8-bits, poniendo el bits más significativo igual a 0 (el de más a la izquierda).

Este código ASCII permite, que equipos fabricados por varios fabricantes, puedan comunicarse porque todos usan este mismo código. La Tabla 2 muestra la relación entre los caracteres ASCII y el valor hexadecimal.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$16	SYN	$2C	, (coma)	$42	B	$58	X	$6E	N
$01	SOH	$17	ETB	$2D	- (dash)	$43	C	$59	Y	$6F	o
$02	STX	$18	CAN	$2E	. (period)	$44	D	$5A	Z	$70	p
$03	ETX	$19	EM	$2F	/	$45	E	$5B	[	$71	q
$04	EOT	$1A	SUB	$30	0	$46	F	$5C	\	$72	r
$05	ENQ	$1B	ESC (escape)	$31	1	$47	G	$5D	]	$73	s
$06	ACK	$1C	FS	$32	2	$48	H	$5E	^	$74	t
$07	BEL (beep)	$1D	GS	$33	3	$49	I	$5F	_ (under)	$75	u
$08	BS (back sp)	$1E	RS	$34	4	$4A	J	$60	`	$76	v
$09	HT (tab)	$1F	US	$35	5	$4B	K	$61	a	$77	w
$0A	LF (linefeed)	$20	SP (space)	$36	6	$4C	L	$62	b	$78	x
$0B	VT	$21	!	$37	7	$4D	M	$63	c	$79	y
$0C	FF	$22	“	$38	8	$4E	N	$64	d	$7A	z
$0D	CR (return)	$23	#	$39	9	$4F	O	$65	e	$7B	{
$0E	SO	$24	$	$3ª	:	$50	P	$66	f	$7C	|
$0F	SI	$25	%	$3B	;	$51	Q	$67	g	$7D	}
$10	DEL	$26	&	$3C	<	$52	R	$68	h	$7E	~
$11	DC1	$27	’ (apost.)	$3D	=	$53	S	$69	i	$7F	DEL (delete)
$12	DC2	$28	(	$3E	>	$54	T	$6A	j		
$13	DC3	$29	)	$3F	?	$55	U	$6B	k		
$14	DC4	$2A	*	$40	@	$56	V	$6C	l		
$15	NAK	$2B	+	$41	A	$57	W	$6D	m		

Tabla 2. Conversión de Hexadecimal a ASCII

Código de trabajo del microprocesador

Los microprocesadores utilizan otro código para dar las instrucciones a la CPU, llamado código de trabajo o ‘**opcode**’. Cada ‘opcode’ instruye a la CPU para ejecutar una sucesión muy específica de pasos, que juntos logran un funcionamiento pretendido. Los microprocesadores de diferentes fabricantes usan diferentes juegos de ‘opcodes’ porque estos ‘opcodes’ son internamente alambrados en la lógica de la CPU. El **juego de instrucciones** para una CPU específica, es el conjunto de todos los funcionamientos que la CPU sabe realizar. Los ‘opcodes’ son una representación del conjunto de instrucciones y los código mnemotécnicos son otros. Aunque los ‘opcodes’ difieren de un microprocesador a otro, todos los microprocesadores realizan los mismos tipos de tareas básicas de maneras similares. Por ejemplo, la CPU MC68HC05 puede entender 62 instrucciones básicas. Algunas de estas instrucciones básicas tienen algunas ligeras variaciones, cada una requiere un ‘opcode’ por separado. El juego de instrucciones del MC68HC05 está representado a través de 210 únicas instrucciones ‘opcodes’. En otro capítulo se muestra cómo la CPU ejecuta realmente las instrucciones, pero primero se necesitan entender unos conceptos más básicos.

Códigos Mnemotécnicos de las Instrucciones y Ensambladores

Un ‘opcode’ como $4C es entendido por la CPU, pero no significa mucho a un humano. Para resolver este problema, se usa un sistema de código mnemotécnico equivalente a la instrucción. El ‘opcode’ $4C corresponde al código mnemotécnico INCA que significa “incrementar el acumulador.” Aunque en este libro se muestra la información que correlaciona las instrucciones mnemotécnicas y los ‘opcodes’ que ellos representan, esta información raramente la usa un programador, porque el proceso de traducción se maneja automáticamente por un programa para el microprocesador llamado **ensamblador** (**assembler**). Un ensamblador es un programa que convierte un programa escrito en código mnemotécnico, en una lista de **código máquina** (‘opcodes’ y otra información) que puede ser usado por una CPU.

Un ingeniero primero desarrolla un conjunto de instrucciones para el microprocesador en forma de código mnemotécnico, después usa un ensamblador para traducir estas instrucciones en 'opcodes' para que la CPU lo pueda entender. En otros capítulos se verán las instrucciones, como escribir un programa y el programa ensamblador. Sin embargo, ahora solo hay que entender que las personas preparan instrucciones para una microprocesador en una forma de código mnemotécnico, pero que el microprocesador entiende sólo 'opcodes'; así que, se requiere un paso, la traducción de cambiar los mnemónicos a 'opcodes', que es la función del ensamblador.

Números Octales

Antes de dejar los sistemas de números y códigos, se ven los dos códigos adicionales de los que se puede haber oído hablar. El sistema **Octal** (**base 8**) se usó para algún trabajo del microprocesador, pero raramente se usa hoy en día. La anotación Octal usa los números del 0 al 7, representados en tres dígitos binarios, lo mismo que en hexadecimal se representan en cuatro dígitos binarios. El sistema octal tiene la ventaja de usar los mismos símbolos de los números decimales, a diferencia de los símbolos hexadecimales, que añaden de la A a la F.

Los dos motivos que causaron el abandono del sistema octal en favor del hexadecimal fueron: En primer lugar, la mayoría de microprocesadores usan palabras de 4, 8, 16, o 32 bits; estas palabras no se pueden agrupar en grupos de tres bits. Algunos de los primeros microprocesadores usaron palabras de 12-bits agrupados en tres bits cada uno. En segundo lugar el sistema octal no es tan compacto como el hexadecimal; por ejemplo, el valor en ASCII para la letra mayúscula A es 1000001_2 (en binario), 41_{16} (en hexadecimal) y 101_8 (en octal). Cuando un programador está hablando sobre el valor ASCII de la letra A, es más fácil decir "cuatro y uno" que decir "uno cero uno".

La Tabla 3 muestra la equivalencia entre el sistema octal y el binario. La columna "binario directo" muestra la equivalencia dígito por dígito, con los dígitos de la columna "octal" en grupos de tres bits binarios. El bit de más a la izquierda (el noveno) se muestra escrito en negrita. Este bit se desecha para hacer los deseados 8-bits. La columna "binario 8-bits" tiene la misma información binaria que la columna "binario directo", exceptuando que los bits se reagrupan en cuatro bits. Cada grupo de cuatro bits equivale a un dígito hexadecimal.

Octal	Binario directo	Binario 8 bits	Hexadecimal
000	**0**00 000 000	0000 0000	$00
001	**0**00 000 001	0000 0001	$01
002	**0**00 000 010	0000 0010	$02
003	**0**00 000 011	0000 0011	$03
004	**0**00 000 100	0000 0100	$04
005	**0**00 000 101	0000 0101	$05
006	**0**00 000 110	0000 0110	$06
007	**0**00 000 111	0000 0111	$07
010	**0**00 001 000	0000 1000	$08
011	**0**00 001 001	0000 1001	$09
012	**0**00 001 010	0000 1010	$0A
013	**0**00 001 011	0000 1011	$0B
014	**0**00 001 100	0000 1100	$0C
015	**0**00 001 101	0000 1101	$0D
016	**0**00 001 110	0000 1110	$0E
017	**0**00 001 111	0000 1111	$0F
101	**0**01 000 001	0100 0001	$41
125	**0**01 010 101	0101 0101	$55
252	**0**10 101 010	1010 1010	$AA
377	**0**11 111 111	1111 1111	$FF

Tabla 3. Equivalencias entre Octal, Binario, y Hexadecimal

Al traducir mentalmente los valores de octal a binario, el valor octal se representa con tres dígitos. Con cada dígito octal representado con grupos de tres bits binarios, hay un bit de más, (3 dígitos x 3 bits = 9 bits). Entonces es fácil olvidarse de quitar el bit de más y terminar con un bit extra (el bit noveno). Al traducir de hexadecimal a binario, es más fácil porque cada dígito hexadecimal equivale exactamente a cuatro bits binarios. Dos dígitos hexadecimales exactamente iguales de ocho bits binarios en un byte.

BCD (Binario Codificado en Decimal)

Binario codificado en decimal (**BCD**) es una anotación híbrida usada para expresar valores decimales en forma binaria, Tabla 4. BCD usa cuatro bits binarios para representar cada dígito decimal. Cuatro dígitos binarios pueden expresar 16 diferentes cantidades físicas, pero en este caso habrá seis combinaciones con valor de bit que son considerados inválidos (específicamente, los valores hexadecimales valor A hasta la F). Los valores BCD se muestran también con el signo $ porque realmente son números hexadecimales que representan cantidades decimales.

Cuando el microprocesador hace que una operación sume en BCD, realiza una suma binaria y entonces ajusta el resultado en la forma BCD. Por ejemplo, la suma BCD siguiente:

$9_{10} + 1_{10} = 10_{10}$

El microprocesador suma

$0000\ 1001_2 + 0000\ 0001_2 = 0000\ 1010_2$

Pero 1010_2 es equivalente a A_{16} que no es un valor BCD válido. Cuando el microprocesador termina el cálculo, se realiza un chequeo para ver si el resultado todavía es un valor BCD válido. Si hubiera cualquier acarreo de un dígito BCD a otro o si hubiera cualquier código inválido, se realizaría una sucesión de pasos para corregir el resultado a la forma BCD apropiada. El binario $0000\ 1010_2$ se corrige a $0001\ 0000_2$ (10 en BCD) en este ejemplo.

Decimal	BCD	Binario	Hexadecimal
0	$0	0000	$0
1	$1	0001	$1
2	$2	0010	$2
3	$3	0011	$3
4	$4	0100	$4
5	$5	0101	$5
6	$6	0110	$6
7	$7	0111	$7
8	$8	1000	$8
9	$9	1001	$9
Combinaciones BCD inválidas		1010	$A
		1011	$B
		1100	$C
		1110	$E
		1111	$F
10	$10	0001 0000	$10
99	$99	1001 1001	$99

Tabla 4. Equivalencia entre Decimal, BCD, Binario y hexadecimal

En la mayoría de casos, no es ineficaz el uso de la anotación BCD en cálculos con microprocesador. Es mejor cambiar de decimal a binario como información entera, el microprocesador hace todos los cálculos en binario y cambia el resultado binario a BCD o a decimal, como se necesite para mostrarlo. Esto es cierto porque: Primero, no todos los microcontroladores son capaces de hacer cálculos en BCD, porque necesitan un indicador de acarreo dígito a dígito que no está presente en todos los microprocesadores (aunque las MCU de Motorola tienen este indicador). Segundo, obliga al microprocesador emular la conducta humana que es menos eficaz que permitir al microprocesador trabajar en su sistema binario nativo.

Resumen

Los microprocesadores tienen dos niveles lógicos (0 y 1) para trabajar con el sistema de numeración binario. Probablemente porque las humanos tienen 10 dedos, se trabaja con el sistema decimal (base 10).

Los números hexadecimales usan 16 símbolos del 0 al 9 y de la A a la F. Cada dígito hexadecimal representa un grupo de cuatro dígitos binarios. El símbolo $ primero se usa como un valor hexadecimal y el símbolo H también como valor hexadecimal para distinguirlo de los números decimales.

ASCII es un código ampliamente aceptado que permite representar la información alfanumérica en valores binarios.

Cada instrucción o variación de una instrucción tiene un único 'opcode' (valor binario) que la CPU reconoce como una petición para realizar una instrucción específica. Las CPUs de diferentes fabricantes tienen juegos diferentes de 'opcodes'.

Los programadores especifican las instrucciones con un código mnemotécnico, como INCA. Un programa llamado ensamblador, traduce el código de instrucciones mnemotécnicas en 'opcodes' para que la CPU las pueda entender.

Elementos básicos de los circuitos Lógicos

Índice

Introducción

Los microprocesadores se componen de elementos digitales lógicos relativamente simples llamados "Puertas", que son pequeños circuitos que pueden conectarse de varias maneras para manejar señales de niveles lógicos. Aunque esto libro no piensa proporcionar información detallada sobre el diseño con lógica, un poco de conocimiento de los elementos lógicos más básicos ayudarán a entender el funcionamiento interno de los microcontroladores.

En este capítulo se empieza con una vista sobre los requisitos de los niveles lógicos de voltaje. También se muestran los transistores y las interconexiones de un microcontrolador CMOS típico. Se explica un simple inversor, una puerta NAND y NOR. Finalmente, se describe una transmisión en una puerta, un 'buffer' de tres estados y un Flip Flop. Virtualmente cualquier parte de un microcontrolador se puede explicar con estos pocos elementos de lógica simple.

Niveles lógicos

Anteriormente, en la discusión de lo que es un microcontrolador, se explicó que un nivel aproximado de 0 voltios indica que es una señal lógica 0 y que un voltaje positivo aproximadamente igual a la fuente de alimentación indica que es una señal lógica 1. Para ser más preciso, hay un nivel de voltaje por debajo del cual el fabricante del microcontrolador garantiza que una señal se reconocerá como válido, lógica 0. Igualmente, hay un nivel de voltaje por encima del cual el fabricante del microcontrolador garantiza que una señal se reconocerá como válido, lógica 1. Al diseñar un sistema con microcontrolador, se asegurará que todas las señales estén conforme a estos límites especificados, incluso bajo las peores condiciones.

La mayoría de microcontroladores modernos usan una tecnología llamada CMOS (Semiconductor del Metal-Óxido Complementario). Esto significa que los circuitos incluyen los dos tipos de transistores el N y el P. Los transistores se explicarán en mayor detalle después de este capítulo.

En un circuito típico CMOS, una entrada lógica 0 puede especificarse entre 0.0 voltios y 0.3 veces V_{DD}. Si V_{DD} es 5.0 voltios, esto traslada el rango de 0.0 a 1.5 voltios. Una entrada lógica 1 puede especificarse entre 0.7 veces la V_{DD} hasta V_{DD}. Si V_{DD} es 5.0 voltios, esto traslada el rango de 3.5 a 5.0 voltios.

Transistores CMOS

La Figura 3 muestra los símbolos para un transistor CMOS tipo N y P. Las características exactas de estos transistores se puede determinar por su esquema físico, tamaño y forma. El propósito de este libro, es tratarlos como dispositivos conmutadores simples.

El transistor tipo N en la Figura 3 tiene su terminal 'fuente' [3] conectado a tierra. Para que un transistor tipo N conduzca, el voltaje de la 'puerta' [2] debe ser más alto que el voltaje de la 'fuente' [3], por una cantidad

conocida como 'umbral'. Se dice que este transistor tipo N conduce (entre los terminales [1] y [3]), cuando hay un voltaje lógico 1 en su 'puerta' [2]. Cuando la 'puerta' está en lógica 0, se dice que este transistor tipo N no conduce y actúa como un interruptor abierto entre los terminales [1] y [3].

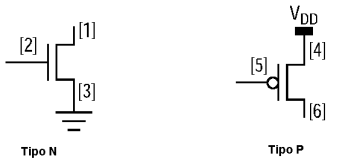


Figura 3. Transistores CMOS N y P

El transistor tipo P en la Figura 3 tiene su terminal 'fuente' [4] conectado a V_{DD}. Para que un transistor tipo P conduzca, su voltaje de 'puerta' [5] debe ser más bajo que el voltaje de la 'fuente' [4], por una cantidad conocida como 'umbral'. El transistor tipo P se diferencia del tipo N con un pequeño circulo en la 'puerta' [5]. Cuando hay un voltaje lógico 0 en la 'puerta' [5] este transistor tipo P se dice que conduce y actúa como un interruptor cerrado entre los terminales [4] y [6]. Cuando la 'puerta' está en voltaje lógico 1, este transistor tipo P no conduce y actúa como un interruptor abierto entre los terminales [4] y [6].

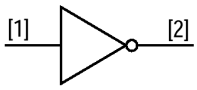
Es relativamente fácil integrar miles transistores del tipo P y N en un microcontrolador y conectarlos de varias maneras para realizar operaciones lógicas complejas. En los siguiente párrafos, se ven algunos de los circuitos lógicos más básicos que se encuentran en un microcontrolador.

Puertas Simples

Los tres tipos de puertas lógicas básicas encontradas en un microcontrolador son el Inversor, la puerta NAND y la puerta NOR. Un diseñador de lógica usa varias combinaciones de estas puertas básicas para formar circuitos lógicos más complejos, como los que suman dos números binarios. Aunque que no se piensa en el libro enseñar técnicas de diseño con lógica, se describen estos circuitos para dar un mejor entendimiento de cómo trabaja un microcontrolador con información digital.

Inversor

La Figura 4 muestra el símbolo lógico de un Inversor, su Tabla de verdad y un circuito CMOS equivalente. Cuando un nivel de señal lógico (0 o 1) se presenta a la entrada [1] de un inversor, el nivel lógico opuesto aparece en su salida [2].



Entrada [1]	Salida [2]
0	1
1	0

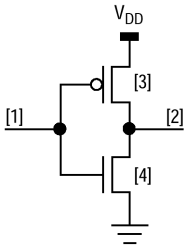


Figura 4. Inversor CMOS

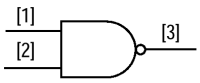
Viendo el circuito CMOS equivalente a la derecha de la Figura 4 y la Tabla 5, se puede explicar lo siguiente: Cuando la entrada [1] está a un nivel lógico 0, el transistor tipo N [4] no conduce y el transistor tipo P [3] conduce, la salida [2] que los une, está a V_{DD} (lógica 1). Cuando la entrada [1] está en nivel lógico 1, el transistor tipo P [3] no conduce y el transistor tipo N [4] conduce, conectando la salida [2] a tierra (lógica 0).

Entrada [1]	Transistor		Salida [2]
	[3]	[4]	
0	On	Off	Conectado a V_{DD} (1)
1	Off	On	Conectado a tierra (0)

Tabla 5. Funcionamiento de una puerta Inversora

Puerta NAND

La Figura 5 muestra el símbolo de una puerta lógica NAND, su Tabla de verdad y un circuito CMOS equivalente. Cuando ambas entradas [1] y [2] de la puerta NAND están en niveles lógicos 1, la salida [3] estará a lógica 0. Si cualquiera de las entradas de la puerta NAND está en lógica 0, la salida estará a 1.



Entrada		Salida
[1]	[2]	[3]
0	0	1
0	1	1
1	0	1
1	1	0

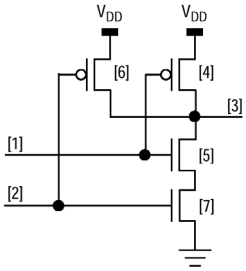


Figura 5. Puerta NAND CMOS

Viendo el circuito CMOS equivalente a la derecha de la Figura 5 y la Tabla 6, se puede explicar lo siguiente: Cuando ambas entradas [1] y [2] están en lógica 1, ninguno de los dos de los transistores tipo P [6] y [4] no conducen y los transistores tipo N [5] y [7] conducen, para que la salida [3] esté conectada a tierra (lógica 0). Cuando la entrada [1] está a lógica 0, el transistor tipo N [5] no conduce y desconecta la salida [3] de tierra, sin tener en cuenta la condición del transistor tipo N [7]. También, cuando la entrada [1] está a un nivel lógico 0, el transistor tipo P [4] conduce, conectando la salida [3] a V_{DD} (lógica 1). Igualmente, cuando la entrada [2] está en nivel lógico 0, el transistor tipo N [7] no conduce, desconectando la salida [3] de tierra, sin tener en cuenta la condición del transistor tipo N [5]. También, cuando la entrada [2] está a un nivel lógico 0, el transistor tipo P [6] conduce, conectando la salida [3] a V_{DD} (lógica 1).

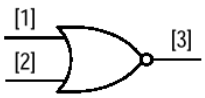
Entrada		Transistor				Salida
[1]	[2]	[6]	[4]	[5]	[7]	[3]
0	0	On	On	Off	Off	V_{DD} (1)
0	1	Off	On	Off	On	V_{DD} (1)
1	0	On	Off	On	Off	V_{DD} (1)
1	1	Off	Off	On	On	GND (0)

Tabla 6. Funcionamiento de la puerta NAND

Aunque ésta es una simple función lógica, muestra cómo los transistores CMOS pueden interconectarse para realizar la lógica Booleana con simples señales de nivel lógico. La lógica de Boole (llamada así, por el matemático irlandés que la formuló) tiene 2 valores (0 y 1) basados en el sistema algebraico con formas matemáticas y relaciones.

Puerta NOR

La Figura 6 muestra el símbolo lógico, la Tabla de verdad y un circuito CMOS equivalente. Cuando ninguna entrada [1] y [2] de una puerta NOR está a un nivel lógico 1, la salida [3] estará a un nivel lógico 1. Si cualquier entrada de la puerta NOR está a un nivel lógico 1, la salida estará a lógica 0.



Entrada		Salida
[1]	[2]	[3]
0	0	1
0	1	0
1	0	0
1	1	0

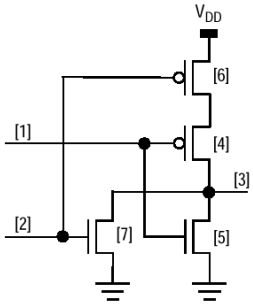


Figura 6. Puerta NOR CMOS

Viendo el circuito CMOS equivalente a la derecha de la Figura 6 y la Tabla 7, se puede explicar lo siguiente: Cuando ambas entradas [1] y [2] están a un nivel lógico 0, los transistores tipo N [5] y [7] no conducen y los transistores tipo P [4] y [6] conducen y la salida [3] queda conectada a V_{DD} (lógica 1). Cuando la

entrada [1] está a un nivel lógico 1, el transistor tipo P [4] no conduce, desconectando la salida [3] de V_{DD} sin tener en cuenta la condición del transistor tipo P [6]. También, cuando la entrada [1] está a un nivel lógico 1, el transistor tipo N [5] conduce, conectando la salida [3] a tierra (lógica 0). Igualmente, cuando la entrada [2] está a un nivel lógico 1, el transistor tipo P [6] no conduce, desconectando la salida [3] de V_{DD} sin tener en cuenta la condición del transistor tipo P [4]. También, cuando la entrada [2] está a lógico 1, el transistor tipo N [7] conduce, conectando la salida [3] a tierra (lógica 0).

Entrada		Transistor				Salida
[1]	[2]	[4]	[5]	[6]	[7]	[3]
0	0	On	Off	On	Off	V_{DD} (1)
0	1	On	Off	Off	On	GND (0)
1	0	Off	On	On	Off	GND (0)
1	1	Off	On	Off	On	GND (0)

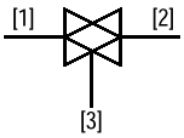
Tabla 7. Tabla de Verdad de la puerta NOR

Puertas de Transmisión, Buffers y Flip-Flops

Los microcontroladores incluyen los tipos más complejos de puertas lógicas y los elementos funcionales mostrados en la sección anterior. En esta sección, se estudian algunas de estas estructuras más complejas, las dos estructuras primeras —la puerta de transmisión y el Buffer de tres estados— introduciendo la idea de señales de alta impedancia controladas lógicamente, la tercera — Medio Flip-Flop — introduce una estructura, que puede mantener una señal en la salida, aún después de que la señal de entrada haya cambiado. Los Flip-Flops son vitales para los microcontroladores, para realizar contadores y tareas secuenciales.

Puerta de transmisión

La Figura 7 muestra el símbolo lógico, la Tabla de verdad y un circuito CMOS equivalente. Cuando la entrada de control [3] está a un nivel lógico 1, se dice que la puerta de transmisión conduce y cualquier nivel lógico presente en la entrada [1] también estará en la salida [2]. Cuando la entrada de control [3] está a un nivel lógico 0, se dice que la puerta de transmisión no conduce y el nodo de salida [2] estará desconectado de todo (alta impedancia o Hi-Z).



Control [3]	Entrada [1]	Salida [2]
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

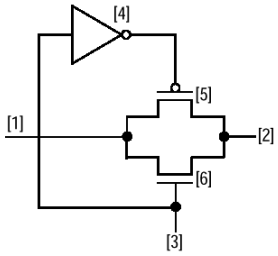


Figura 7. Puerta de Transmisión CMOS

Viendo el circuito CMOS equivalente a la derecha de la Figura 6 y la Tabla 7, se puede explicar lo siguiente: Cuando la entrada de control [3] está a un nivel lógico 0, la puerta del transistor tipo N [6] estará a lógico 0 y la puerta del transistor tipo P [5] estará a un nivel lógico 1 (V_{DD}). No hay voltaje entre tierra y VDD, que harían conducir el transistor tipo P [5] o el transistor tipo N [6], así que no hay ninguna conducción entre la entrada [1] y la salida [2]. Desde el nodo de salida [2] está eficazmente aislado de todo, entonces se dice que está en alta impedancia (Hi-Z).

Cuando la entrada de control [3] está a un nivel lógico 1, se dice que la puerta de transmisión conduce y entonces tiene una conexión directa de la entrada [1] a la salida [2]. Si las dos entradas de control [3] y [1] están a un nivel lógico 1, el transistor tipo P [5] conducirá y conectará la entrada [1] y salida [2]. Aunque la puerta de transistor tipo N [6] esté en nivel lógico 1, la 'fuente' [1] también está al mismo voltaje, entonces el transistor [6] no conducirá. Si la entrada de control [3] está a un nivel lógico 1 y la entrada [1] está a un nivel lógico 0, el transistor tipo N [6] conducirá y conectará la entrada [1] y la salida [2]. Aunque la puerta del transistor tipo P [5] esté a un nivel lógico 0, la 'fuente' [1] también estará al mismo voltaje, entonces el transistor [5] no conducirá.

La puerta de transmisión mostrada en la Figura 7 a veces se llama interruptor analógico porque es capaz de dejar pasar señales intermedias entre los niveles lógicos legales.

Las puertas de transmisión pueden formar multiplexores de datos, como el mostrado en la Figura 8. Cuando la señal seleccionada [3] está a un nivel lógico 1, la puerta de transmisión [6] conducirá y la puerta de transmisión [7] (debido al inversor [5]) no conducirá. Así que la salida [4] no tendrá el mismo nivel lógico que la entrada [1] y las señales 1 de la entrada [2] no afectarán a la salida [4]. Cuando la señal seleccionada [3] está a un nivel lógico 0, la puerta de transmisión [7] conducirá y la puerta de transmisión [6] no conducirá. Así que la salida [4] no tendrá el mismo nivel lógico que la entrada [2] y las señales 1 en la entrada [1] no afectarán a la salida.

Selección [3]	Entrada		Salida [4]
	[1]	[2]	
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

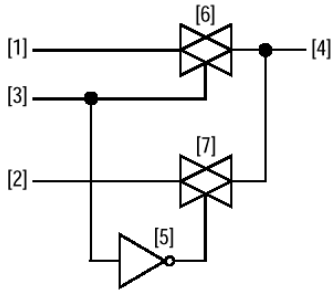


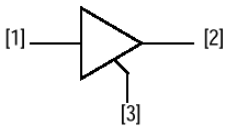
Figura 8. Multiplexor de Datos 2:1

Selección	1	2	6	7	4
0	0	0	Off	On	0
0	0	1	Off	On	1
0	1	0	Off	On	0
0	1	1	Off	On	1
1	0	0	On	Off	0
1	0	1	On	Off	0
1	1	0	On	Off	1
1	1	1	On	Off	1

Tabla 8. Funcionamiento del Multiplexor de Datos

Buffer de tres estados

La Figura 9 muestra el símbolo lógico, un circuito CMOS equivalente y la Tabla de verdad para un Buffer de tres estados. Cuando la entrada de control [3] está a un nivel lógico 0, se dice que el Buffer no conduce y la salida [2] está en alta impedancia que aísla el nodo. Cuando la entrada de control [3] está a un nivel lógico 1, se dice que el Buffer conduce y cualquier nivel lógico que esté presente en la entrada [1] también estará en la salida [2].



Control [3]	Entrada [1]	Salida [2]
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

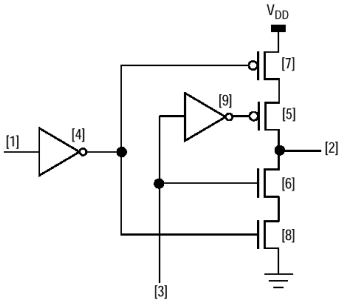


Figura 9. Buffer de Tres-estados

Viendo el circuito CMOS equivalente a la derecha de la Figura 9 y la Tabla 9, se puede explicar lo siguiente: Cuando la entrada de control [3] está a un nivel lógico 0, la puerta del transistor tipo N [6] estará a un nivel lógico 0 y la puerta del transistor tipo P [5] a través del inversor [9] estará a un nivel lógico 1 (V_{DD}), entonces ambos transistores [5] y [6] no conducirán. El nodo de salida [2] quedará aislado del todo, se dice que está en alta impedancia.

Cuando la entrada de control [3] está a un nivel lógico 1, la puerta del transistor tipo N [6] estará a un nivel lógico 1 y la puerta del transistor tipo P [5] estará a un nivel lógico 0. Si la entrada del Buffer [1] está a un

nivel lógico 0, la salida del inversor [4] estará a un nivel lógico 1 y hará conducir el transistor tipo N [8] y dejará de conducir el transistor tipo P [7]. Con la entrada de control [3] a un nivel lógico 1 y la entrada [1] a un nivel lógico 0, la salida del Buffer [2] estará conectada a tierra a través de los transistores tipo N [6] y [8] qué estarán en conducción.

Cuando la entrada de control [3] está a un nivel lógico 1, la puerta del transistor tipo N [6] estará a un nivel lógico 1 y la puerta del transistor tipo P [5] estará a un nivel lógico 0. Si la entrada del Buffer [1] está a un nivel lógico 1, la salida del inversor [4] estará a un nivel lógico 0, haciendo conducir el transistor tipo P [7] y dejará de conducir el transistor tipo N [8]. Con la entrada de control [3] y la entrada [1], ambas a un nivel lógico 1, la salida del Buffer [2] se conectará a V_{DD} a través de los transistores tipo P [7] y [5], que estarán en conducción.

Control [3]	Entrada [1]	Nodo		Transistor				Salida [2]
		[4]	[9]	[5]	[6]	[7]	[8]	
0	0	1	1	Off	Off	Off	On	Hi-Z
0	1	0	1	Off	Off	On	Off	Hi-Z
1	0	1	0	On	On	Off	On	GND (0)
1	1	0	0	On	On	On	Off	V_{DD} (1)

Tabla 9. Funcionamiento del Buffer

Medio Flip Flop (HFF)

La Figura 10 muestra el símbolo lógico y un circuito CMOS equivalente de un medio Flip-Flop (HFF). Cuando la entrada de reloj [2] está a un nivel lógico 1, la puerta de transmisión [9] conduce y la puerta de transmisión [8] no conduce. Se dice que el medio Flip-Flop es transparente, porque la señal de entrada [1] pasa directamente a las salidas Q [3] y Q negada [4]. Cuando el reloj [2] está a un nivel lógico 0, la puerta de transmisión [8] conduce y la puerta de transmisión [9] no conduce. En este estado, el medio Flip-Flop se dice que está enclavado (latched). La puerta de transmisión [8], el inversor [6] y el inversor [7] forman un "anillo" estable y las salidas Q [3] y Q negada [4] permanecen al mismo nivel lógico que cuando el reloj cambió de 1 a 0.

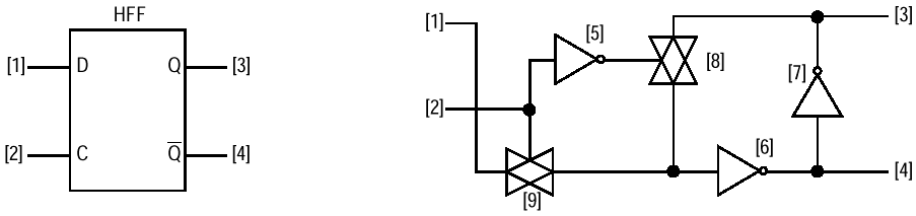


Figura 10. Medio Flip Flop

Resumen

Aunque se piensa a menudo que los niveles lógicos son 0 voltios o 5 voltios, estos son realmente rangos de voltaje garantizados por el fabricante de la MCU. Para una MCU específica que trabaja con V_{DD} = 5.0 V, un nivel lógico 0 puede ser de 0.0 a 1.5 V y un nivel lógico 1 puede ser de 3.5 a 5.0 V. Siempre hay que referirse a las hojas de datos para cada MCU para obtener los voltajes lógicos 0 y 1.

Las MCU CMOS se componen de miles de transistores tipo N y P. Un transistor tipo N conduce (conduce de la 'fuente' hasta el 'drenador') cuando su 'puerta' está a un nivel lógico 1 y su 'fuente' está a un nivel lógico 0. Un transistor P conduce cuando su fuente está a un nivel lógico 1 y su puerta está a un nivel lógico 0.

Los transistores N y P pueden conectarse de varias maneras para realizar un trabajo lógico. Hay tres tipos de puertas simples: los Inversores, las puertas NAND y las puertas NOR. La salida de un inversor siempre tiene un nivel lógico opuesto a su nivel de entrada. La salida de una puerta NAND está en nivel lógico 0 cuando todas sus entradas están en nivel lógico 1. La salida de una puerta NOR está a un nivel lógico 0 cuando cualquiera o todas sus entradas están a un nivel lógico 1.

La salida de una puerta de transmisión o un Buffer de tres estados puede tener nivel lógico 0, lógico 1 o alta impedancia. Un salida tiene alta impedancia cuando parece no estar conectada a nada (un circuito abierto).

Un medio Flip-Flop (HFF) tiene una condición transparente y una condición de enclavamiento (latched). En la condición de transparente (con la entrada de reloj igual a un nivel lógico 1), la salida Q es siempre igual al nivel de la lógica presentada a la entrada. En la condición de enclavamiento (la entrada de reloj igual a un nivel lógico 0), la salida mantiene el nivel lógico que fue presente cuando el Flip-Flop estaba en su último lugar en la condición de transparente. Los cambios en el nivel lógico de entrada, mientras el Flip-Flop está enclavado, no afectan al nivel lógico de salida.

Memoria y Entradas/Salidas Paralelas

Índice

Introducción

Antes de ver el funcionamiento de la CPU se puede discutir en detalle algunos conocimientos conceptuales que requiere la memoria del microcontrolador. En muchas clases de inicio a la programación, la memoria se presenta como algo similar a unas casillas donde se pueden almacenar mensajes y otras informaciones. Las casillas en que aquí se van a referir son como los buzones de un gran edificio. Es una buena analogía, pero necesita un poco más de detalle para explicar los funcionamientos internos de una CPU.

Analogía de una Memoria con unas Casillas

La idea de que cualquier tipo de memoria sirve para guardar información, es cierta, pero no hay ningún punto que diga como se guarda dicha información y no se tiene de una manera fiable como se puede recuperar dicha información cuando se necesita. La serie de buzones de un edificio puede usarse como un tipo de almacenamiento de memoria. Se puede poner información en un buzón numerado y cuando se quiere recuperar dicha información, se puede ir al buzón con esa dirección y recuperarla.

A continuación, se puede llevar de manera análoga para explicar simplemente cómo ve la memoria un microcontrolador de 8-bits para que se pueda ser más específico. En una CPU de 8-bits, cada casilla (o buzón) puede ser un contenedor de un conjunto de ocho interruptores (on/off). En una casilla real, no se le puede poner más información escribiendo más pequeño, ni tampoco hay ninguna casilla vacía (son ocho interruptores y son On o Off). Los contenidos de una posición de memoria pueden ser desconocidos o indefinidos a un momento dado, así mismo, los interruptores en las casillas pueden estar en un estado desconocido hasta que sean definidos por primera vez. Los ocho interruptores estarán en una fila, donde cada interruptor representa un solo dígito binario (bit). Un 1 binario corresponde a un interruptor cerrado (On) y un 0 binario corresponde a un interruptor que está abierto (Off). Cada casilla (posición de memoria) tiene una única dirección para que la información pueda guardarse y recuperarse fiablemente.

En un edificio, las direcciones de los buzones pueden ser del 100 al 175 para el primer piso, del 200 al 275 para el segundo piso, etc. Éstos son números decimales que tienen significando para las personas. Anteriormente se dijo que las microcontroladores trabajan con el sistema numérico binario y un microcontrolador con cuatro líneas de dirección puede identificar 16 direcciones. Este microcontrolador identifica las direcciones de las 16 posiciones de memoria (buzones) con el valor hexadecimal de $0 a $F.

Los microcontroladores MC68HC05 más pequeños, tienen 10 líneas de dirección que permiten direccionar 1024 únicas posiciones de memoria. En comparación, el microcontrolador MC68HC11 de 8-bits tiene 16 líneas de dirección que pueden direccionar 65,536 únicas posiciones de memoria.

Cómo ve la Memoria el Microprocesador

Una CPU de 8-bits con 10 líneas de dirección ve la memoria como una fila continua de 1024 valores de 8-bits. La primera posición de memoria tiene la dirección 00 0000 0000_2 y la última posición tiene la dirección 11 1111 1111_2. Estos 10-bits de direcciones normalmente se expresan como dos números de 8-bits, esto se expresa a su vez por cuatro dígitos hexadecimales. En la anotación hexadecimal, estas direcciones van de $0000 a $03FF.

El microcontrolador especifica qué posición de memoria se está accediendo (leer desde o escribir a) poniéndose una única combinación de 1 y 0 en las 10 líneas de dirección. La intención para leer la posición o escribir en la posición se hace poniendo un 1 (lectura) o un 0 (escritura) en una línea llamada read/write (R/W). La información 'de' o 'para' la posición de memoria se lleva con ocho líneas de datos. En cualquier posición de memoria de un microcontrolador se puede 'escribir a' o se puede 'leer de'. No todos los tipos de memoria se pueden escribir, pero es trabajo del programador saber esto, no el microcontrolador. Si un programador ordena erróneamente al microcontrolador escribir en una memoria de sólo lectura, el microcontrolador lo intentará.

Kilobytes, Megabytes y Gigabytes

La unidad más pequeña de memoria de un microcontrolador es de un solo **bit** que puede guardar un valor 0 o 1. Estos bits se agrupan en conjuntos de ocho bits, para hacer un **byte**. Los microcontroladores más grandes se agrupan en grupos más extensos de 16 o 32 bits, para hacer una unidad llamada **palabra** o **word**. El tamaño de una palabra puede ser diferente para diferentes microcontroladores.

En el mundo decimal, se expresan a veces números muy pequeños o muy grandes, incluyendo un prefijo como mili, kilo, etc., antes de la unidad de medida. En el mundo binario, se acostumbra a utilizar prefijos similares para describir grandes cantidades de memoria. En el sistema decimal, el prefijo kilo significa 1000 (o 10^3). En el sistema binario, la potencia del número 2 que más se acerca a 1000_{10} es $2^{10} = 1024_{10}$. Cuando se dice kilobytes (Kbytes) son múltiplos de 1024_{10} bytes y aunque esta terminología científica no es del todo exacta, a través de los años de uso se ha quedado como normal.

Un megabyte es 2^{20} o $1{,}048{,}576_{10}$ bytes. Un gigabyte es 2^{30} o $1{,}073{,}741{,}824_{10}$ bytes. Un microcontrolador personal con 32 líneas de dirección, teóricamente puede direccionar 4 gigabytes ($4{,}294{,}967{,}296_{10}$) de memoria. Los microcontroladores pequeños discutidos en este libro tienen desde 512 bytes hasta 16 kilobytes de memoria.

Tipos de Memoria

Los microcontroladores utilizan varios tipos de información que requiere almacenarla en diferentes tipos de memoria. Las instrucciones que controlan el funcionamiento de los microcontroladores se guardan en una memoria **no-volátil** para que el sistema no tenga que ser reprogramado después de que se deje de alimentar. Para trabajar con las variables y los resultados intermedios, es necesario guardarse en una memoria que pueda escribirse rápidamente y fácilmente durante el funcionamiento del sistema. No es importante conservar este tipo de información cuando no hay alimentación, por lo que puede usarse una memoria **volátil**. Estos tipos de memorias se les puede modificar su contenido escribiendo y sólo se pueden leer por la CPU del microcontrolador.

Otra información sobre las memorias, los datos de entrada son leídos por la CPU y los datos de salida son escritos por la CPU. Las entradas/salidas E/S (I/O input/output) y los registros de control son también una forma de memoria del microcontrolador, pero son diferentes a los otros tipos de memoria, porque la información puede ser detectada y/o cambiada por otra cosa diferente de la CPU.

Memorias Volátiles

RAM: Memoria de Acceso Aleatorio (Random Access Memory)

La RAM es una forma de memoria volátil que puede ser leída o escrita por la CPU. Como su nombre implica, se puede acceder a las posiciones de la RAM en cualquier orden. Esta es el tipo de memoria más común en un microcontrolador. La RAM requiere una cantidad relativamente grande de área de silicio del circuito integrado y debido a esto el costo es alto, por esto, normalmente las cantidades de RAM incluidas en los microcontroladores suelen ser pequeñas.

Memorias No-Volátiles

ROM: Memoria de sólo Lectura (Read-Only Memory)

La ROM consigue su información durante el proceso industrial del circuito integrado. La información debe ser proporcionada por el cliente antes de integrar el circuito, que una vez fabricado contendrá esta información. Cuando se ha terminado el microcontrolador, esta información puede ser leída por la CPU pero no puede cambiarse. La ROM está considerada como una memoria no-volátil porque la información no cambia si se deja de alimentarla. La ROM es el tipo más simple, más pequeña y más barata de memoria no-volátil.

PROM: Memoria Programable de sólo lectura (Programmable ROM)

La PROM es similar a la ROM, sólo que puede se programa después de fabricar el circuito integrado.

EPROM: Memoria Eléctricamente Programable (borrable por UV)

La EPROM puede ser borrada exponiéndola a una fuente de luz ultravioleta. Los microcontroladores con EPROM tienen una pequeña ventana de cuarzo que permite al circuito integrado ser expuesto a la luz ultravioleta para borrarse. El número de veces que puede borrarse y reprogramarse una EPROM se limita a unos cientos ciclos y depende del dispositivo en particular. Para programar la información en una memoria EPROM se usa un procedimiento especial. La mayoría de microcontroladores con EPROM usan un voltaje adicional típico de +12 Vdc, durante la programación de la EPROM. La CPU no puede escribir información en una posición de la EPROM tal como se hace para escribir en una posición de la RAM.

Algunos microcontroladores tienen internamente un circuito de programación de la EPROM, para que la CPU del microcontrolador pueda programar las posiciones de memoria de la EPROM. Cuando la EPROM se está programando, no se conectan los buses de direcciones y de datos tal como se haría en una memoria normal. De manera análoga a las casillas, esto sería quitar el mueble entero de soporte de los buzones y tomándolo como un almacén donde las cajas se llenarían con información. Mientras los buzones están programándose, no se puede acceder a los buzones del edificio. Algunos microcontroladores con EPROM (no el MC68HC705J1A) tienen un modo especial de funcionamiento que los hace parecer una memoria EPROM normal de la industria. Estos dispositivos pueden programarse con un programador de propósito general de EPROM.

OTP: Memoria EPROM programable uno sola vez

Cuando un microcontrolador con EPROM se encapsula con plástico opaco, se le llama un microcontrolador de programable una sola vez o OTP. Como que la luz ultravioleta no puede atravesar el encapsulado, la memoria no puede borrarse. El silicio dentro de una OTP es idéntico a uno encapsulado con una ventana de cuarzo. El encapsulado en plástico es mucho más barato que el encapsulado en cerámico con una ventana de cuarzo. Las MCU con OTP son ideales para aplicaciones de alto volumen.

EEPROM: Memoria Programable y Borrable eléctricamente

Las EEPROM se pueden borrar eléctricamente por los comandos de un microcontrolador. Para programar un nuevo valor en una posición, se debe borrar primero dicha posición y entonces realizar una serie de pasos de programación. Esto es algo más complicado que cambiar una posición de la RAM a la que simplemente puede escribirse un nuevo valor por la CPU. La ventaja de la EEPROM es que es una memoria no-volátil. La EEPROM no pierde sus contenidos cuando se deja de alimentarla. La diferencia con la Memoria RAM es el número de veces que se puede borrar y reprogramar una posición de EEPROM que está limitada, típicamente a 10,000 ciclos. El número de veces que se puede leer una posición de EEPROM es ilimitado.

FLASH: Memoria Programable y Borrable eléctricamente

Las memorias Flash se pueden borrar y programar eléctricamente (típicamente a 10.000 ciclos de programación/borrado). Ocupan menor área de silicio dentro de un circuito integrado.

E/S como un tipo de Memoria

El estado y el control de las Entradas/Salidas, es un tipo de posición de memoria que permite al sistema microcontrolador conseguir la información ‘a’ o ‘del’ mundo exterior. Este tipo de posición de memoria es inusual porque la información puede detectarse y/o puede cambiarse por otra cosa diferente a la CPU.

Los tipos más simples de posiciones de memoria son los puertos de entrada y salida. En una MCU de 8-bits, un simple puerto de entrada consiste en ocho pins que pueden ser leídos por la CPU. Un simple puerto de salida consiste en ocho pins que la CPU puede controlar (escribir a). En la práctica, una posición de un simple puerto de salida normalmente se lleva a cabo con ocho básculas y ocho caminos de realimentación, que permiten a la CPU leer lo que se escribió previamente en la dirección de dicho puerto de salida.

La Figura 11 muestra los circuitos equivalentes de un bit de RAM, un bit de un puerto de entrada y un bit de un puerto de salida típico que tienen la capacidad de almacenar la lectura anterior. En una MCU real, estos circuitos se repiten ocho veces para hacer una sola posición de 8-bits de RAM, puerto de entrada o puerto de salida. El medio flip-flop (HFF), Figura 11, es transparente a la entrada. Cuando la señal de reloj está en estado alto, los datos pasan libremente de la entrada D a la salida Q y 1. Cuando la entrada de reloj está en estado bajo, los datos se almacenan en las salidas Q y 1.

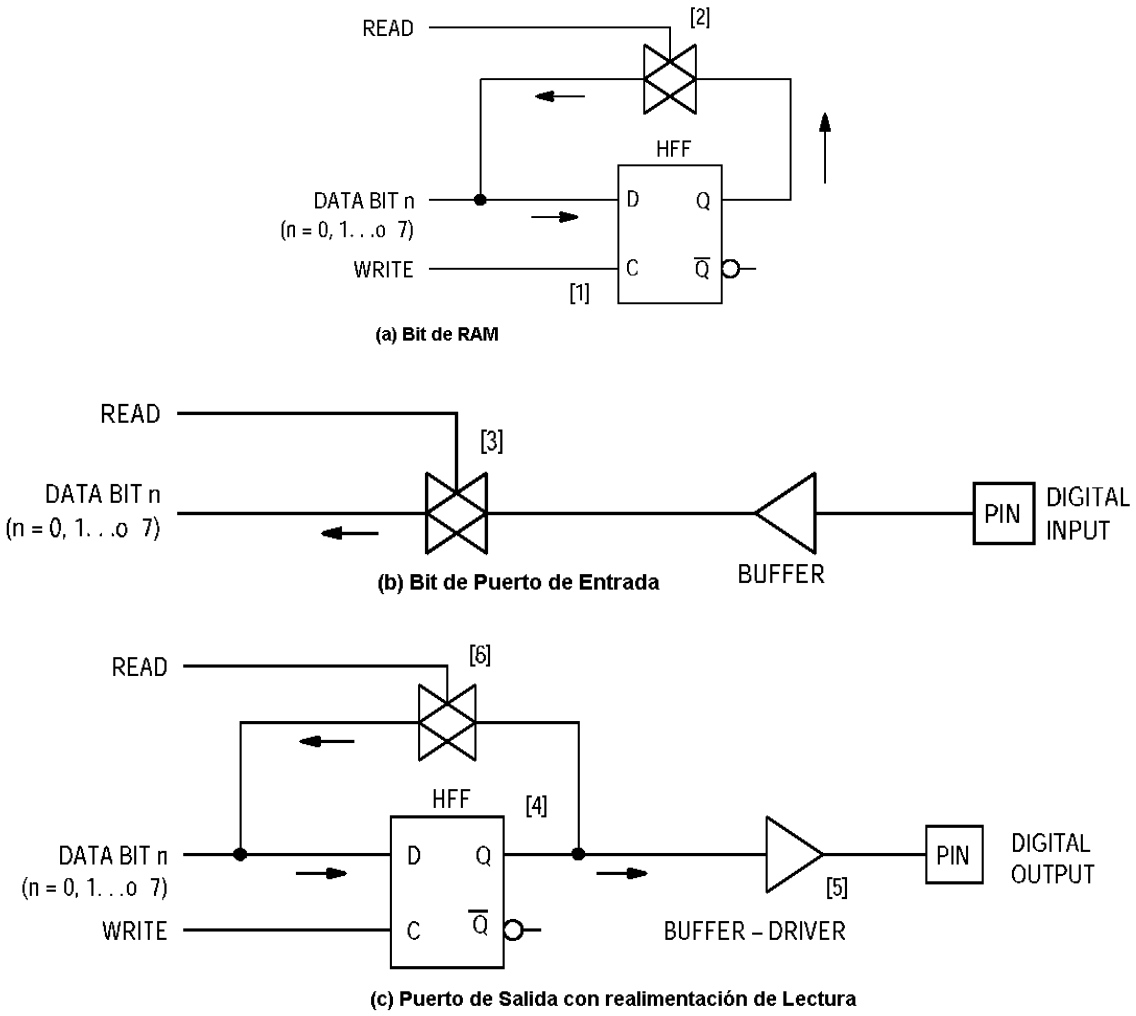


Figura 11. Memoria y Circuitería de E/S

Cuando la CPU guarda un valor en la dirección que corresponde el puerto de salida en la Figura 11 (c), se activa la señal WRITE para almacenar el dato de la línea del bus de datos en el medio flip-flop [4]. La salida de esta bascula, la cual tiene un buffer [5], aparece como un nivel digital en el terminal de salida. Cuando la CPU lee la dirección de este puerto de salida, se activa la señal READ para habilitar el multiplexor [6]. Este multiplexor acopla los datos de salida del medio flip-flop hacia la línea bus de datos.

Estados internos y Registros de control

Los estados internos y los registros de control son simplemente posiciones de memoria. En lugar de detectar y controlar los pines externos, el estado y los registros de control detectan los niveles de señal lógicos internos (indicadores).

Mirando la Figura 11 y comparando el bit de una RAM con el puerto de salida. La única diferencia es que el bit de salida tiene un buffer para conectar el medio flip-flop a un pin externo. En el caso de un bit de control interno, la salida del buffer se conecta a alguna señal de control interno en lugar de a un pin externo. Un bit de estado interno está como un bit de puerto de entrada excepto que la señal que es detectada durante una lectura, es un señal interna en lugar de un pin externo.

La familia de microcontroladores MC68HC05 incluyen pines de E/S paralelos. La dirección de cada terminal es programable por un software accesible por un bit de control. La Figura 12 muestra la lógica para un pin de E/S bidireccional que incluye un latch en el puerto de salida y un bit de control de dirección de datos.

Un pin del puerto se configura como una salida si su correspondiente bit DDR (registro de dirección de datos) se pone a 1. Un pin se configura como una entrada si su correspondiente bit DDR se pone a 0. Un 'reset' o 'power on', ponen a 0 todos los bits DDR y configura todos los pines del puerto como entradas. Los bits DDR son capaces de ser escritos o leídos por el procesador.

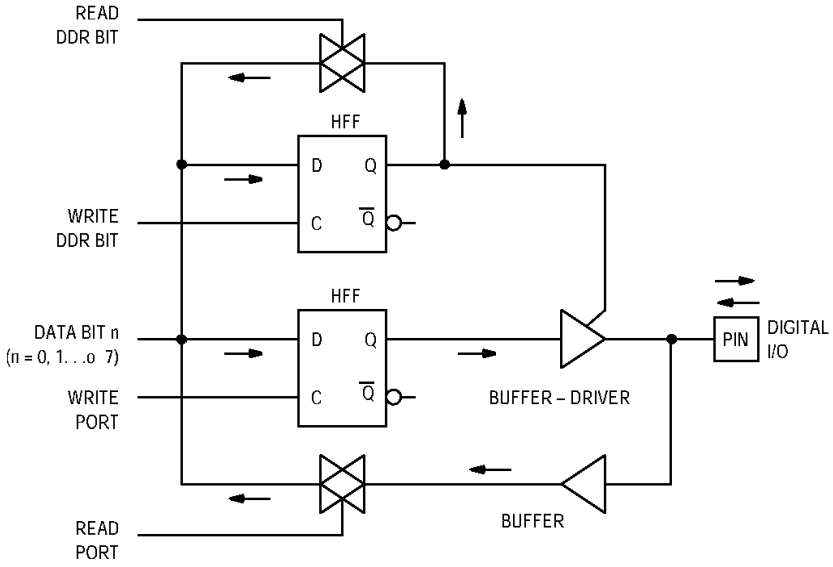


Figura 12. Puerto de E/S con Control de Dirección de Datos

Mapa de memoria

Puesto que hay más de mil posiciones de memoria en un sistema MCU, es importante saber de alguna manera por donde se ha pasado. Un **mapa de memoria** es una representación gráfica del espacio total de la memoria de la MCU. La Figura 14 es un mapa de memoria típico que muestra los recursos de la memoria del MC68HC705J1A.

A lo largo de la columna izquierda de la Figura 14 se muestran las direcciones, con los valores de 4-dígitos hexadecimales empezando por arriba con $0000 y aumentando hasta $07FF en la parte inferior. $0000 corresponde a la primera posición de memoria (seleccionada cuando la CPU tiene todas las líneas del bus interno de direcciones a 0). $07FF corresponde a la última posición de memoria seleccionada (cuando la CPU tiene todas las 11 líneas del bus interno de direcciones a 1). La columna siguiente, dentro de rectángulos, se identifican los tipos de memorias (RAM, EPROM, registros de E/S, etc. , que residen en una área particular de memoria). Cada rectángulo puede interpretarse como una fila de 2048 casillas (posiciones de memoria) y cada una de estas contiene ocho bits de datos como lo muestra la Figura 13.

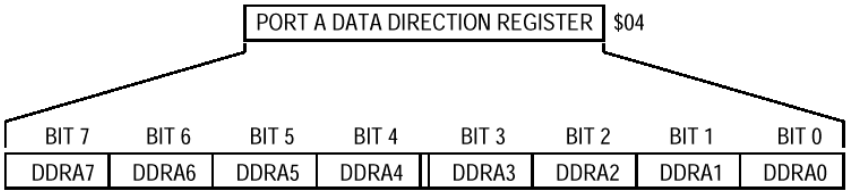


Figura 13. Detalle expandido de una posición de memoria

Algunas áreas, como los registros de E/S, necesitan ser mostradas en más detalle porque es importante saber los nombres de cada posición individual.

A las primeras 256 posiciones de memoria ($0000–$00FF) se puede acceder por el microcontrolador de un modo especial llamado 'modo de direccionamiento directo'. Los modos de direccionamiento se describen en mayor detalle en el **Juego de Instrucciones del MC68HC05**.

En el modo de direccionamiento directo, la CPU asume que los dos dígitos hexadecimales superiores de la dirección son 0; así, en la instrucción sólo se necesita dar explícitamente los dos dígitos de la parte baja de la dirección. Los registros internos de E/S y los 64 bytes de RAM se localizan en el área de memoria $0000–$00FF.

En el mapa de memoria de la Figura 14, la expansión del área de memoria de E/S identifica cada posición del registro con los dos dígitos de la parte baja de su dirección en lugar de los 4-dígitos completos de la dirección.

Por ejemplo, los valores de los 2-dígitos hexadecimales $00 aparecen a la derecha del registro de datos del puerto A, qué realmente se localiza en la dirección $0000 del mapa de memoria.

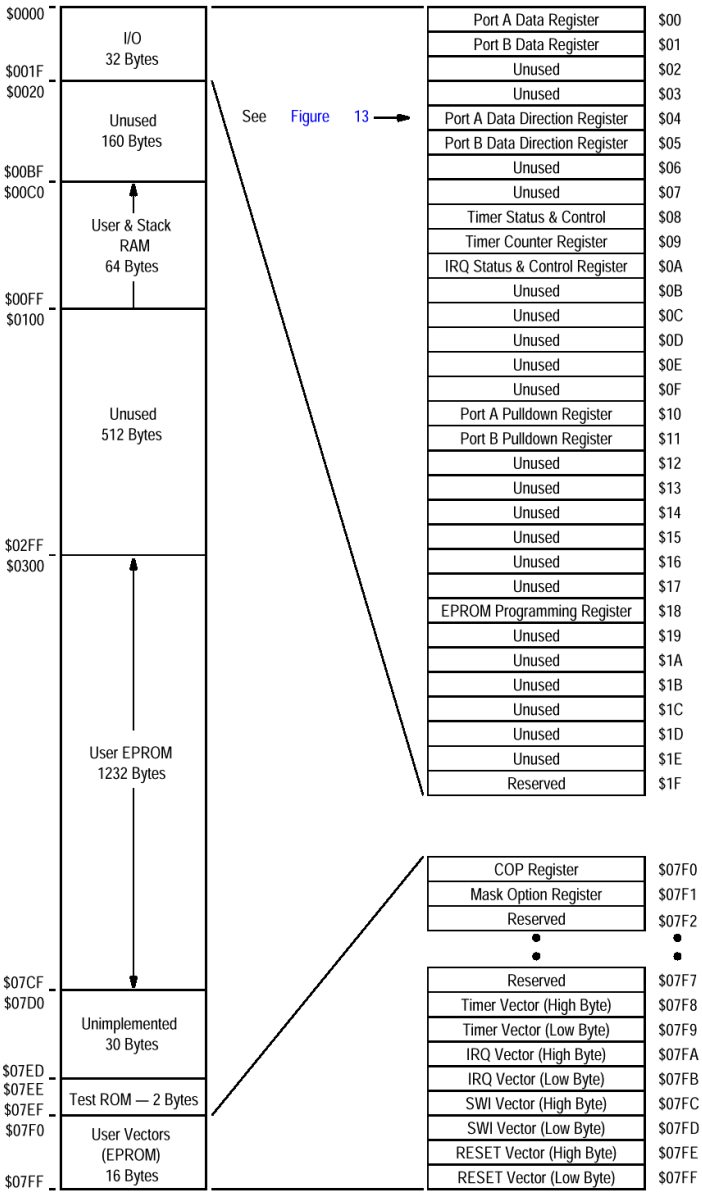


Figura 14. Mapa de Memoria Típico

La Memoria como un Periférico

Las memorias pueden tener una forma de periférico. Los usos de los diferentes tipos de memoria fueron discutidos anteriormente, pero no fue considerada la circuitería lógica que se requiere para soportar éstas memorias. Las memorias ROM y RAM son circuitos directos, no requieren ningún soporte lógico que la de selección de dirección, para distinguir una posición de otra. Esta circuitería lógica de selección está contenida en el mismo circuito de la propia memoria.

Las memorias EPROM (PROM borrable) y EEPROM (PROM eléctricamente borrable) requieren una lógica de soporte para la programación (y de borrado en el caso de la EEPROM). Por ejemplo, la circuitería lógica de soporte, de los periféricos en el MC68HC705J1A, contiene internamente un programador de PROM en la MCU. Un registro de control incluye unos bits de control para seleccionar entre los modos de programación y de lectura, para habilitar el alto voltaje alto de alimentación requerido para la programación.

Resumen

Se puede pensar que una memoria del microcontrolador es como una serie de buzones, pero un microcontrolador ve la memoria como una serie valores de 8-bits.

Si un microcontrolador tiene *n* líneas de dirección, únicamente puede direccionar 2^n posiciones de memoria. Un microcontrolador con 11 líneas de dirección pueden direccionar 2^{11}, o 2048_{10} posiciones.

NOTA: Un kilobyte (escrito 1 Kbyte) es igual a 1024_{10} bytes.

Tipos de Memoria

- RAM— Memoria de acceso aleatorio, puede leerse o puede escribirse por una CPU. Los contenidos son guardados mientras se le aplica alimentación.
- ROM— Memoria de sólo lectura, puede leerse pero no puede modificarse. Los contenidos deben determinarse antes de que sea fabricado el circuito integrado. No se requiere alimentación para guardar sus contenidos.
- EPROM— Memoria ROM programable y borrable, puede cambiarse el contenido borrándola con luz ultravioleta y programándola con un nuevo valor. Los ciclos de borrado y programación están limitados por un número de veces. No requiere alimentación para guardar sus contenidos. Contiene una pequeña ventana de cuarzo, para poder acceder la luz ultravioleta y borrarla.
- OTP — El chip interno es idéntico al de una EPROM, pero es programable una sola vez y está encapsulado en plástico opaco. Puesto que la luz ultravioleta no puede atravesar el plástico, la memoria no puede borrarse después de que haya sido programada.
- EEPROM — Memoria PROM borrable eléctricamente, puede cambiarse el contenido por medio de señales eléctricas y guardar sus contenidos cuando no se aplica alimentación. Típicamente, una posición de EEPROM puede borrarse y reprogramarse 10,000 veces.
- FLASH — Memoria borrable y programable eléctricamente por bloques.
- E/S— Registros de E/S, de control y de estado son un tipo especial de memoria, porque la información puede detectarse y/o cambiarse por otra cosa diferente de la CPU.
- Memoria no-volátil— La memoria No-volátil guarda sus contenidos cuando no hay alimentación.
- Memoria volátil— La memoria Volátil pierde sus contenidos cuando se deja de alimentar.

NOTA: **Mapa de memoria** — Un mapa de memoria es una vista gráfica de todas las posiciones de memoria en un sistema microcontrolador.

A las primeras 256 posiciones de un sistema microcontrolador se pueden acceder de una manera especial, llamado modo de direccionamiento directo. En el modo de direccionamiento directo, la CPU asume que el byte de la parte alta de la dirección es $00 para que no se tenga que ser dado explícitamente en un programa (ahorrando el espacio que habría tomado y eliminando el ciclo de reloj que habría necesitado).

Especialmente se pueden considerar las memorias EPROM y EEPROM como periféricos en un sistema microcontrolador. La circuitería de soporte y de control de programación es necesaria para modificar los contenidos de estas memorias. Esto difiere de las simples memorias RAM que pueden leerse o escribirse en un solo ciclo de reloj de la CPU.

Arquitectura del Microcontrolador

Índice

Introducción

Este capítulo toca el corazón del microcontrolador para ver como trabaja. Será una mirada más detallada que normalmente necesita el usuario, pero ayudará a entender, por qué algunas cosas se hacen de una cierta manera.

Todo lo que hace la CPU se basa en la secuencia de pasos simple. Por ejemplo, un oscilador genera un reloj que es usado para hace funcionar la CPU a través de estas secuencias. El reloj de la CPU es muy rápido, hablando en términos humanos parece que las cosas están pasando casi instantáneamente. Pasando por estas secuencias paso a paso, se entenderá el funcionamiento de cómo un microcontrolador ejecuta los programas. También se tomará conocimiento de las capacidades de un microcontrolador, así como sus limitaciones.

Arquitectura del Microcontrolador

Los microcontroladores de Motorola MC68HC05 y MC68HC11 de 8-bits tienen una organización específica, que se llama arquitectura Von Neumann, nombre de un matemático americano. En esta arquitectura, la CPU y una serie de memoria está interconectada por un bus de direcciones y un bus de datos. El **bus de direcciones** identifica la posición de memoria que se está accediendo y el **bus de datos** se usa para llevar la información de la CPU a la **posición de memoria** (casilla) o de la posición de memoria a la CPU.

Motorola en la implementación de esta arquitectura, tiene unas casillas especiales dentro de la CPU, llamadas registros de la CPU, las cuales actúan como un pequeño bloc de notas (llamado en inglés: small scratch pad) y como un panel de control para la CPU. Estos registros de la CPU son similares a los de una memoria, en que la información se puede escribir en ellos y almacenarlos. Sin embargo, es importante recordar que estos registros están alambrados directamente en la CPU y no forma parte de la memoria direccionable disponible en la CPU.

Toda la información (otra cosa que los registros de la CPU) accesible a la CPU está prevista (por la CPU) estar en una sola fila de mil o más casillas. Esta organización a veces se la llama sistema de **mapa de memoria** de **E/S** porque la CPU trata todas las posiciones de memoria igual como si ellas contienen instrucciones de programa, **variables** de datos, o controles de **entrada-salida** (E/S). Hay otras arquitecturas de microcontrolador, pero en este libro no está pensado explicar estas variaciones.

Afortunadamente, la arquitectura de la familia MC68HC05 de Motorola es más fácil de entender y usar. Esta arquitectura abarca las ideas más importantes de los microcontroladores binarios digitales; así que, la información presentada en este libro es aplicable a otras arquitecturas.

El número de líneas en el bus de direcciones determina el número total de casillas posibles; el número de líneas en el bus de datos determina la cantidad de información que puede guardarse en cada casilla. El MC68HC705J1A, por ejemplo, el bus de direcciones tiene 11 líneas, teniendo un máximo de 2048 casillas (en general se dice que la MCU puede acceder a 2 K posiciones). El bus de datos del MC68HC705J1A es de ocho bits, cada casilla puede soportar un byte de información. Un byte tiene ocho dígitos binarios o dos dígitos hexadecimales o un carácter ASCII o un valor decimal de 0 a 255.

Registros de la CPU

Las diferentes MCU tienen diferentes juegos de registros de la CPU. Las diferencias son principalmente en el número y el tamaño de los registros. La Figura 15 muestra los registros de la CPU de la familia MC68HC05. Aunque la CPU tiene un juego de registros relativamente simples, son representativos de todos los tipos de registros de CPU y pueden ser usados para explicar todos los conceptos fundamentales. Este capítulo proporciona una descripción breve de los registros del MC68HC05, como una introducción a la arquitectura de las CPU en general. En el **Juego de Instrucciones del MC68HC05** se incluye información más detallada sobre los registros del MC68HC05.

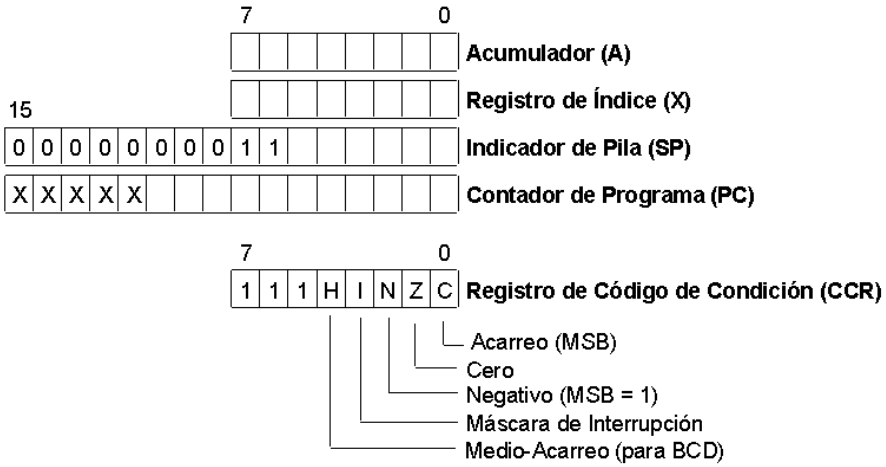


Figura 15. Registros de la CPU de MC68HC05

El **registro (A)** es un registro 'scratch pad' de 8-bits, también llamado **acumulador** porque a menudo se usa para almacenar uno de los operandos o el resultado de una operación aritmética.

El **registro (X)** es un registro de índice de 8-bits, que también puede servir como un simple 'scratch pad'. El propósito principal del registro de índice es apuntar a una área de memoria donde la CPU carga (lee) o guarda (escribe) la información. A veces un registro de índice se le llama **registro puntero**. Se aprenderá más sobre los registros de índice cuando se describa el modo de direccionamiento indexado.

El **registro contador de programa (PC)** es usado por la CPU para guardar la pista de la dirección de la siguiente instrucción a ser ejecutada. Cuando se alimenta la CPU o después de un 'reset', el registro PC se carga desde un par de posiciones específicas de memoria llamadas **vector de reset**. Las posiciones del vector de reset contienen la dirección de la primera instrucción que será ejecutada por la CPU. Así que las instrucciones se ejecutan, la lógica de la CPU incrementa el registro PC, tal que siempre apunta al próximo fragmento de información que la CPU necesitará. El mismo número de bits que adapta el registro PC, será exactamente igual al número de líneas del bus de direcciones. Esto determina el espacio total de memoria potencialmente disponible que puede acceder la CPU. En el caso del MC68HC705J1A, el registro PC es de 11 bits; por consiguiente, su CPU puede acceder a 2 Kbytes (2048 bytes) de memoria. Los valores para este registro se

expresan con cuatro dígitos hexadecimales, donde los cinco bit de la parte superior de los 16 bits correspondientes a la dirección binaria, siempre estarán a 0.

El **registro de código de condición (CCR)** es un registro de 8-bits y guarda los indicadores de estado, que reflejan el resultado de alguna operación anterior de la CPU. Los tres bits de la parte alta de este registro no se usan y siempre son iguales a 1. Las **instrucciones Branch** (bifurcación) usan los bits de estado para hacer simples tomas de decisiones.

El registro **indicador de pila (SP)** se usa como un indicador de la siguiente posición disponible en una pila, cuya característica es: *la última que entra, es la primera que sale* (LIFO o Last In Firts Out). La pila o '**stack'** puede pensarse que es como un montón de tarjetas, cada una de ellas guarda un solo byte de información. En cualquier momento dado, la CPU puede poner una tarjeta encima de la pila o puede tomar una tarjeta de la pila. Las tarjetas dentro de la pila no pueden recogerse a menos que todas las tarjetas amontonadas encima se hayan quitado primero. La CPU logra este efecto de la pila por medio del registro SP. El registro SP apunta a una posición de memoria (casilla), igual que si se piensa en la siguiente tarjeta disponible. Cuando la CPU envía un fragmento de dato hacia la pila, el valor del dato se escribe en la casilla apuntada por el registro SP, entonces el SP se decrementa para apuntar a la siguiente posición de memoria (casilla) anterior. Cuando la CPU extrae un dato de la pila, el registro SP se incrementa para apuntar a la casilla recientemente usada y el valor del dato se leerá de esa casilla. Cuando la CPU se pone en marcha o después de una instrucción 'reset del indicador de pila' (RSP), el SP apunta a una posición de memoria específica de la RAM (una cierta casilla).

Tiempos

Se usa un **reloj (clock)** de alta frecuencia (típicamente derivada de un cristal de cuarzo conectado a la MCU), para controlar las secuencias de las instrucciones de la CPU. Las MCU típicas dividen la frecuencia básica del cristal por dos o más para llegar a la frecuencia de reloj de bus. Cada memoria lee o escribe a uno ciclo de frecuencia de reloj del bus. En el caso del MC68HC705J1A, se puede trabajar con un oscilador de 4 MHz (máximo), para dividir por dos y llegar a un reloj de 2 MHz (máximo) del procesador interno. Cada subpaso de una instrucción necesita un ciclo de este reloj de bus interno (500 ns). La mayoría de las instrucciones necesitan de dos a cinco de estos subpasos; así que, la CPU es capaz de ejecutar más de 500,000 instrucciones cada segundo.

Vista de un Programa

El **Listado 1. Programa Ejemplo**, es un listado de un pequeño programa de ejemplo, que se usa en la descripción de la CPU. En el capítulo de programación se proporciona información detallada de cómo se escribe un programa. Un listado de un programa proporciona mucha más información que la CPU necesita, porque los humanos también necesitan leer y entender los programas. La primera columna de la izquierda del listado, muestra las direcciones con cuatro dígitos hexadecimales. La siguiente columna muestra los valores de 8-bits (los contenidos de las posiciones de memoria individuales). El resto de la información del listado es la realmente necesaria para el programador, que necesita leer el listado. El significado de toda esta información se describirá en mayor detalle en el capítulo titulado **Programación**.

La Figura 16 muestra un mapa de memoria del MC68HC705J1A y donde está situado el programa de ejemplo en la memoria de la MCU. Esta figura es la misma que la Figura 14 excepto que se ha extendido una porción diferente del espacio de memoria para mostrar los contenidos de todas las posiciones del programa de ejemplo. La Figura 16 muestra que la CPU ve, el programa de ejemplo, como una sucesión lineal de códigos binarios, incluyendo las instrucciones y los **operandos**, en posiciones de memoria sucesivas. Un operando es otro valor del ''opcode'' que la CPU necesita para completar la instrucción. La CPU empieza este programa con su contador de programa (PC) apuntando al primer byte del programa. Cada 'opcode' de la instrucción le dice a la CPU qué tipo y cuántos operandos (si es que tiene alguno) van con esa instrucción. De esta manera, la CPU puede permanecer alineada a los límites de la instrucción, aunque la mezcla de ''opcodes' y de operandos pueden confundir.

La mayoría **programas de aplicación** se localizarán en la ROM, EPROM o OTP, aunque no hay ningún requisito especial para que las instrucciones se deban ejecutar en una memoria del tipo ROM. Por lo que concierne a la CPU, cualquier programa es una serie de patrones de bits binarios, que son secuencialmente procesados.

Listado 1. Programa Ejemplo

```
*********************************************************
* Programa ejemplo 68HC05
* Lee el estado de un interruptor en el bit 0 del puerto A; 1 = cerrado
* Cuando se cierra, el LED se enciende 1 segundo; El LED se enciende
* cuando el bit 7 del Puerto A es 0. Espera a que se abra el interruptor,
* y entonces repetir. Rebotes del interruptor de 50mS
* NOTA: Tiempos basados en el tiempo de ejecución de las instrucciones
* Si usa un simulador o cristal menor de 4MHz, esta rutina correrá más
* lenta que la deseada
*********************************************************
$BASE 10T          ; Díce al ensamblador que usa el modo decimal
                   ; menos los valores $ o %
0000         PORTA EQU $00      ;Dirección Directa del puerto A
0004         DDRA  EQU $04      ;Dirección control de dato, puerto A
00E0         TEMP1 EQU $C0      ;One byte temp storage location
0300               ORG $0300    ;El Programa empezará en $0300
0300 A6 80   INIT  LDA #$80     ;Empieza la inicialización
0302 B7 00         STA PORTA    ;El LED se apagará
0304 B7 04         STA DDRA     ;Pone el bit 7 del puerto como salida
* El resto del puerto A se configura como entrada
0306 B6 00   TOP   LDA PORTA    ;Lee el interruptor en LSB del Puerto A
0308 A4 01         AND #$01     ;Prueba el bit-0
030A 27 FA         BEQ TOP      ;Lazo hasta bit-0 = 1
030C CD 0323       JSR DLY50    ;Retardo de 50 ms para los rebotes
030F 1F 00         BCLR 7,PORTA ;Enciende el LED (bit-7 a 0)
0311 A6 14         LDA #20      ;El Decimal 20 se ensambla como $14
0313 CD 0323       DLYLP JSR DLY50 ;Retardo de 50 ms
0316 4A            DECA         ;Contador de Lazo hasta 20 lazos
0317 26 FA         BNE DLYLP    ;20 veces (20-19,19-18,...1-0)
0319 1E 00         BSET 7,PORTA ;Apaga el LED
031B 00 00FD OFFLP BRSET 0,PORTA,OFFLP ;Lazo hasta interruptor abierto
031E CD 0323       JSR DLY50    ;Retorno realizado
0321 20 E3         BRA TOP      ;Mira al siguiente cierre del interruptor

***
* DLY50 — Subrutina de retardo ~50ms
* Guarda el valor original del acumulador
* pero X siempre estará a cero en el retorno
***
0323 B7 C0 DLY50   STA TEMP1    ;Guarda el acumulador en la RAM
0325 A6 41         LDA #65      ;lazo externo de 65 veces
0327 5F    OUTLP   CLRX         ;X se usa como lazo interno contador
0328 5A    INNRLP  DECX         ;0-FF, FF-FE,...1-0 256 lazos
0329 26 FD         BNE INNRLP   ;6 ciclos * 256 * 500ns/ciclo = 0.768ms
032B 4A            DECA         ;65-64, 64-63,...1-0
032C 26 F9         BNE OUTLP    ;1545 ciclos * 65 * 500ns/ciclo = 50.212ms
032E B6 C0         LDA TEMP1    ;Recupera el valor guardado en el acumulador
0330 81            RTS          ;Retorno
```

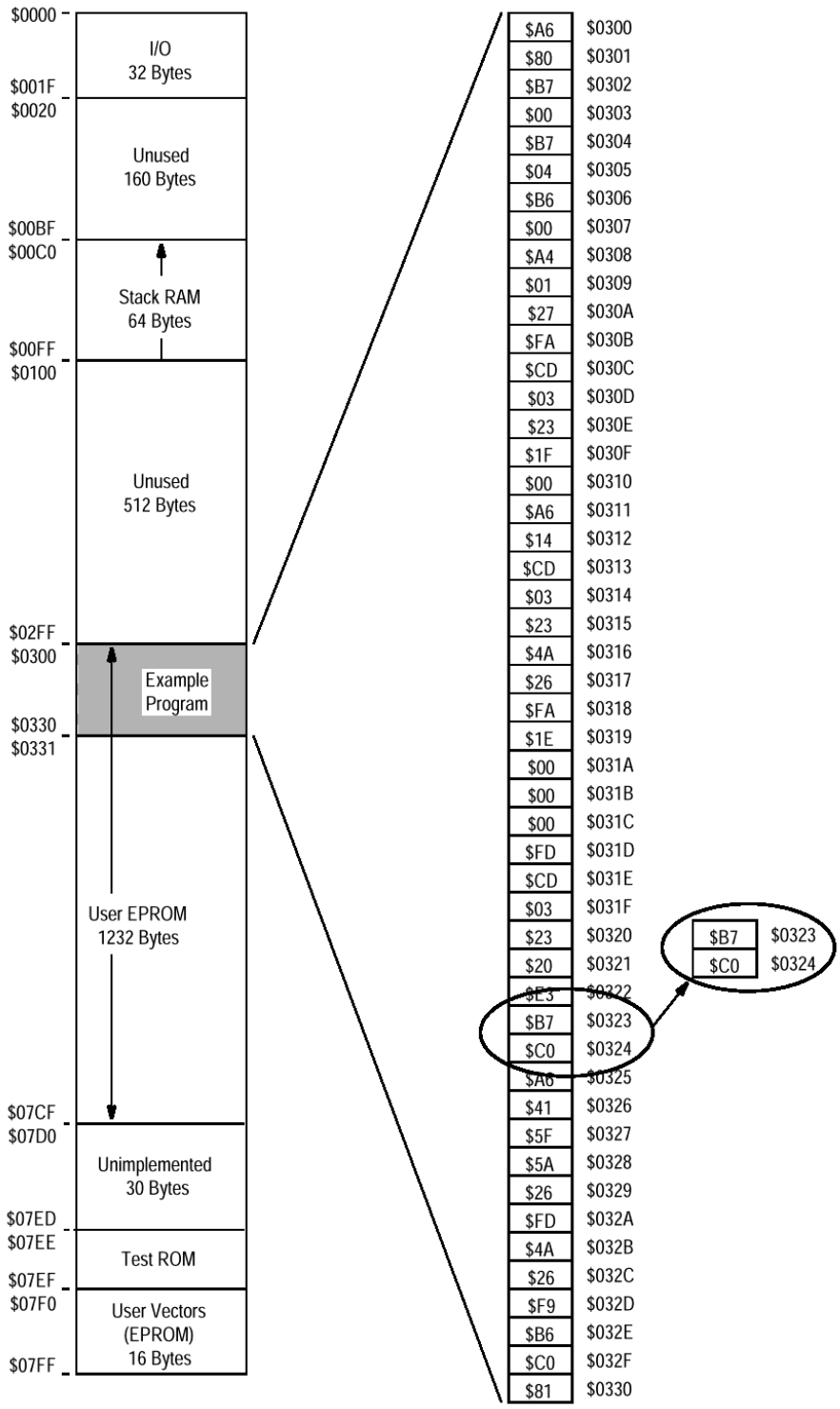


Figura 16. Mapa de Memoria del Programa Ejemplo

Se puede estudiar cuidadosamente el programa de ejemplo en el Listado 1 y el mapa de memoria en la Figura 16. Se debe encontrar la línea que se muestra a continuación, en el **Listado 1. Programa Ejemplo,** donde empieza la primera instrucción de la subrutina DLY50 con sus dos bytes B7 C0.

```
0323    B7 C0          DLY50          STA    TEMP1  ;Guardar el acumulador en la RAM
```

También en la Figura 16 se pueden ver resaltados los dos bytes.

Funcionamiento de la CPU

Esta sección, primero describe el funcionamiento detallado de las instrucciones de la CPU y explica cómo la CPU ejecuta el programa ejemplo. La descripción detallada de las instrucciones típicas de la CPU hacen ver como funciona una CPU. A través del programa ejemplo y usando la técnica de aprendizaje llamada "ver funcionando el microcontrolador", se pretende interpretar la CPU, así como ejecutando las instrucciones de un programa.

Funcionamiento Detallado de las Instrucciones en la CPU

Antes de ver cómo la CPU ejecuta los programas, ayudaría a conocer (en detalle) cómo las micro instrucciones operan en el interior de la CPU, realizando el trabajo fundamental y ejecutando estos pasos diminutos para lograr una instrucción deseada. Como se verá, se ejecutan rápidamente muchos pasos pequeños y con precisión dentro de cada instrucción, pero ninguno de los pasos pequeños es demasiado complicado.

La circuitería lógica dentro de la CPU parece sencilla para un ingeniero de diseño acostumbrado a trabajar con lógica TTL o incluso con lógica de relés. Pero para realizar una MCU resultaría un circuito muy grande. Con las técnicas de muy alta integración (VLSI = Very Large Scale Integration) se ha hecho posible encajar el equivalente a miles de circuitos integrados TTL en un sencillo chip de silicio. Poner en orden estas puertas lógicas, para formar una CPU, puede ser muy complejo y lo mejor es entender la CPU como **caja negra** y dedicarse a programar. Poniendo combinaciones de diferentes instrucciones en el dispositivo, se puede realiza virtualmente cualquier función definible.

Una instrucción típica necesita de dos a cinco ciclos del reloj interno del procesador. Aunque no es normalmente importante saber exactamente que pasa durante cada uno de éstos ciclos de ejecución, ayudará mucho, ir en detalle a través de unas cuantas instrucciones, para entender cómo trabaja internamente la CPU.

Guardar en el Acumulador (Modo de Direccionamiento Directo)

Buscando la instrucción STA en el capítulo **Juego de Instrucciones**, en la tabla se puede ver que $B7 es el modo de direccionamiento directo (DIR) de la instrucción ‘guardar en el acumulador’. También se puede ver que la instrucción requiere dos bytes, uno para especificar el ‘opcode’ ($B7) y el segundo para especificar la **dirección directa** donde será guardado el acumulador. (Los dos bytes se muestran como *B7 dd* en la columna de código máquina de la tabla.)

Los modos de direccionamiento se verán en más detalle en otro capítulo, pero la siguiente descripción ayudará a entender cómo la CPU ejecuta esta instrucción. En el modo de direccionamiento directo, la CPU asume que la dirección está en el rango de $0000 a $00FF; así que no se necesita incluir el byte superior de la dirección del operando en la instrucción (siempre es $00).

En la misma tabla STA muestra que el modo de direccionamiento directo de la instrucción STA necesita cuatro ciclos de reloj de la CPU. Durante el primer ciclo, la CPU pone el valor del contador de programa en el bus de direcciones interno y lee el ‘opcode’ $B7, qué identifica la instrucción como de direccionamiento directo de la instrucción STA y avanza el PC a la siguiente posición de memoria.

Durante el segundo ciclo, la CPU pone el valor del PC en el bus de direcciones interno y lee el byte de la parte baja de la dirección directa (por ejemplo $00). La CPU usa el tercer ciclo de esta instrucción STA para construir internamente la dirección completa donde el acumulador será guardado y avanza el PC para apuntar a la siguiente dirección de memoria (la dirección del ‘opcode’ de la siguiente instrucción).

En este ejemplo, la CPU añade el valor asumido $00 (debido al modo de direccionamiento directo) al $00 que leyó durante el segundo ciclo de la instrucción, para llegar a la dirección completa $0000. Durante el cuarto ciclo de esta instrucción, la CPU pone esta dirección construida ($0000) en el bus de direcciones interno, una vez puesto el valor del acumulador en el bus de datos interno, se declara la señal ‘write’ (escribe). Es decir, la CPU escribe los contenidos del acumulador a $0000 durante el cuarto ciclo de la instrucción STA.

Mientras que el acumulador es guardado, los bits N y Z del registro de código de condición estarán a 1 o 0, según los datos que se guardaron. La formula lógica Booleana o resultado de estos bits para una instrucción determinada aparece en una columna del juego de instrucciones. El bit Z se pondrá a 1, si el valor guardado fue $00; por otro lado, el bit Z se pondrá a 0. El bit N se pondrá a 1, si el bit más significativo del valor guardado fue 1; por otro lado, el bit N se pondrá a 0.

Cargar el Acumulador (Modo de Direccionamiento Inmediato)

A continuación, se puede buscar la instrucción LDA en el capítulo **Juego de Instrucciones**. El modo de direccionamiento inmediato (IMM) de esta instrucción aparece como *A6 ii* en la columna de código máquina de la tabla. Esta instrucción necesita dos ciclos reloj del procesador interno para ser ejecutada.

El 'opcode' $A6 le dice a la CPU que coja el byte de datos que sigue inmediatamente al 'opcode' y que ponga este valor en el acumulador. Durante el primer ciclo de esta instrucción, la CPU lee el 'opcode' $A6 y avanza el PC para apuntar a la siguiente posición de memoria (la dirección del operando inmediato *ii*). Durante el segundo ciclo de la instrucción, la CPU lee los contenidos del byte que siguen al 'opcode' en el acumulador y avanza el PC para apuntar a la siguiente posición de memoria (por ejemplo, el byte del 'opcode' de la siguiente instrucción).

Mientras el acumulador se estaba cargando, los bits N y Z del registro de código de condición fue puesto a 0 o a 1, según los datos que estaba cargando en el acumulador. La formula lógica Booleana o resultado de estos bits para una instrucción determinada aparece en una columna del juego de instrucciones. El bit Z se pondrá a 1 si el valor cargado en el acumulador fue $00; por otro lado, el bit Z se pondrá a 0. El bit N se pondrá a 1 si el bit más significativo del valor cargado fue 1; por otro lado, el bit N se pondrá a 0.

El bit de código de condición N (negativo) se puede usar para detectar el signo de un número **complemento a dos**. En números complemento a dos, el bit más significativo se usa como un bit de signo, 1 indica un valor negativo y 0 indica un valor positivo. El bit N también se puede usar como un simple indicador de estado del bit más significativo de un valor binario.

Bifurcación Condicional

La instrucción 'branch' (bifurcación) permite a la CPU seleccionar uno de los dos caminos del flujo de programa, dependiendo del estado de un bit en particular de la memoria o varios bits de código de condición. Si la condición detectada por la instrucción 'branch' es verdadera, el flujo del programa salta a una posición especificada de la memoria. Si la condición detectada no es verdadera, la CPU continúa la instrucción que sigue a la instrucción 'branch'. La decisión en un diagrama de flujo bloquea las instrucciones de bifurcación condicional en el programa.

La mayoría de instrucciones 'branch' contienen dos bytes, un byte para el 'opcode' y otro byte para el desplazamiento relativo. La instrucción 'branch' con bit a 0 (BRCLR) y 'branch' con bit a 1 (BRSET) requieren tres bytes: un byte para el 'opcode', un byte de direccionamiento directo (para especificar la posición de memoria a ser probada) y otro byte de desplazamiento relativo.

El byte de desplazamiento relativo es interpretado por la CPU como un valor complemento a dos con signo. Si la condición 'branch' verificada es verdadera, el desplazamiento se agrega al PC y la CPU lee la siguiente instrucción de esta nueva dirección calculada. Si la condición 'branch' no es verdadera, la CPU simplemente continúa a la siguiente instrucción después de la instrucción 'branch'.

Llamada a una subrutina y Retorno de una subrutina

Las instrucciones de salto a subrutina (JSR = jump to subrutine) y bifurcación a subrutina (BSR = branch to subrutine) automatizan el proceso de dejar la línea de flujo normal del programa para irse y ejecutar un conjunto de instrucciones y después volver al flujo normal de donde se había salido. Al conjunto de instrucciones fuera del flujo normal del programa se llama subrutina. Una instrucción JSR o BSR se usa para ir del programa normal a la subrutina. Para volver de una subrutina se usa una instrucción de retorno de subrutina (RTS), para volver al programa del que la subrutina fue llamada.

El **listado 2. Ejemplo de Llamada a Subrutina**, muestra un listado en ensamblador que se puede usar para ver cómo la CPU ejecuta una llamada a una subrutina. Hay que asumir que el indicador de la pila (SP) apunta a la dirección $00FF cuando la CPU encuentra la instrucción JSR en la posición $0302. Los listados en ensamblador se describen detalladamente en el capítulo titulado **Programando**.

Listado 2. Ejemplo de Llamada a Subrutina

```
  "      "    "                      "                          "
0300     A6  02          TOP         LDA     #$02        ;Carga un valor inmediato
0302     CD 04 00                    JSR     SUBBY       ;Salto a subrutina
0305     B7  E0                      STA     $E0         ;Guarda el acumulador en la RAM
0307      "    "                      "                          "
  "      "    "                      "                          "
  "      "    "                      "                          "
0400     4A              SUBBY DECA                      ;Decrementa el acumulador
0401     26  FD                      BNE SUBBY           ;Lazo hasta acumulador = 0
0403     81                          RTS                 ;Vuelve al programa principal
```

Para la descripción siguiente se puede ver la Figura 17. Se empieza con la ejecución de la instrucción LDA #$02 en la dirección $0300. En el lado izquierdo de la figura se muestra el flujo normal del programa compuesto por TOP LDA #$02, JSR SUBBY y STA $E0 (en este orden) en posiciones de memoria consecutivas. En el lado derecho de la figura se muestran las instrucciones de la subrutina SUBBY DECA, BNE SUBBY y RTS.

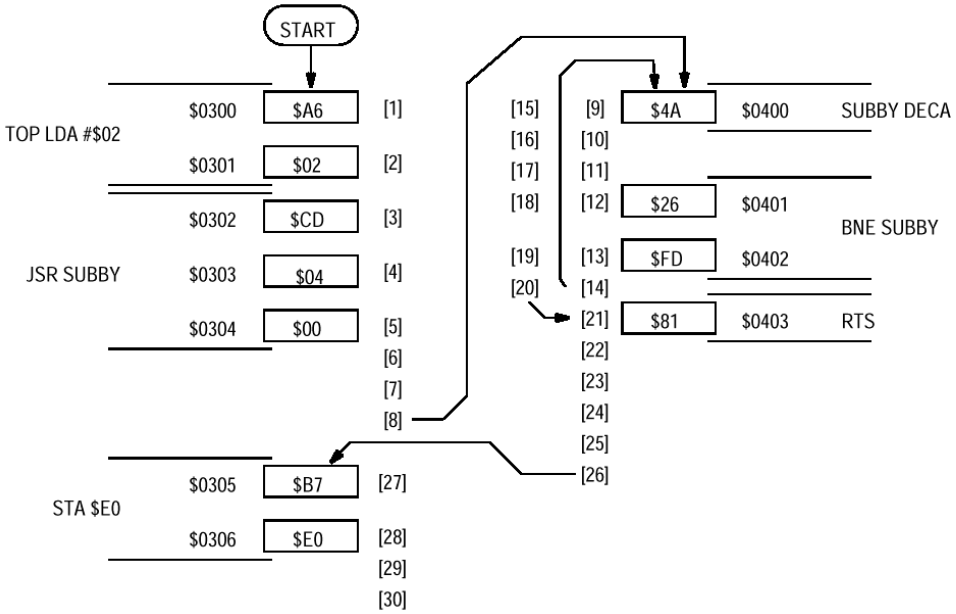


Figura 17. Secuencia de Llamada a Subrutina

El número de ciclos de reloj de la CPU entre corchetes de la Figura 17, se usan como referencia para la descripción siguiente.

[1] La CPU lee el 'opcode' $A6 de la posición $0300 (LDA inmediato).
[2] La CPU lee inmediatamente el dato $02 de la posición $0301al acumulador.
[3] La CPU lee el 'opcode' $CD de la posición $0302 (JSR extendido).
[4] La CPU lee la parte alta de la dirección extendida $04 de $0303.
[5] La CPU lee la parte baja de la dirección extendida $00 de $0304.
[6] La CPU contruye la dirección completa de la subrutina ($0400).
[7] La CPU escribe $05 en $00FF y decrementa el SP a $00FE. Otra manera de decir esto es: "envía la parte baja de la dirección de retorno a la pila (stack)".
[8] La CPU escribe $03 en $00FE y decrementa el SP a $00FD. Otra manera de decir esto es: "envía la parte alta de la dirección de retorno a la pila". La dirección de retorno en que se guardó la pila es $0305, que es la dirección de la instrucción que sigue a la instrucción JSR.
[9] La CPU lee el 'opcode' $4A de la posición $0400. Ésta es la primera instrucción de la llamada a subrutina.
[10] La CPU usa la ALU para substraer un valor en el acumulador.

[11] El resultado de la ALU (A–1) se escribe al acumulador.
[12] La CPU lee el 'opcode' ($26) BNE de la posición $0401.
[13] La CPU lee el **desplazamiento relativo** ($FD) de $0402.
[14] Durante la instrucción LDA #$02 a [1], el acumulador fue cargado con el valor 2; durante la instrucción DECA a [9], el acumulador era decrementado a 1 (qué no es igual a 0). Así que, a [14], la condición de 'branch' era verdadera y el desplazamiento de complemento a dos ($FD o –3) se agregó al PC interno (qué en ese momento era $0403) para conseguir el valor $0400.
[15] a [19] se repiten los ciclos de [9] a [13] excepto que cuando la instrucción DECA [15] se ejecutó en este momento, el acumulador pasó de $01 a $00.
[20] Puesto que ahora el acumulador es igual a 0, la condición de bifurcación BNE [19] no es verdadera y no se tomará la bifurcación.
[21] La CPU lee el 'opcode' ($81) RTS de $0403.
[22] Incrementa el SP a $00FE.
[23] Lee parte alta de la dirección de retorno ($03) de la pila.
[24] Incrementa el SP a $00FF.
[25] Lee la parte baja de la dirección de retorno ($05) de la pila.
[26] Se recupera la dirección completa $0305 y se guarda en el PC.
[27] La CPU lee el 'opcode' ($B7) STA directo de la posición $0305.
[28] La CPU lee la parte baja de la dirección directa ($E0) de la posición $0306.
[29][30] La instrucción directa STA necesita un total de cuatro ciclos. Durante los últimos dos ciclos de la instrucción, la CPU construye la dirección completa donde el acumulador será guardado, añadiendo $00 (valor supuesto para la parte alta de la dirección, debido al modo de direccionamiento directo) al $E0 leído durante [28]. El acumulador ($00 en este momento) se guarda en la dirección construida ($00E0).

Ver Funcionando un Microcontrolador

Ver funcionando un microcontrolador, es el mejor ejercicio de aprendizaje donde se pretende ser una CPU que ejecuta un programa. Los programadores a menudo prueban mentalmente un programa para ver funcionar un microcontrolador como si leyeran a través de una rutina de software. Mientras que al ver funcionando un microcontrolador, no es necesario aislar instrucciones por ciclos individuales del procesador. En cambio, una instrucción se trata como una sola operación completa en lugar de pasos individuales para realizar dicha instrucción.

Los párrafos siguientes muestran el proceso de funcionamiento de un microcontrolador, yendo a través del ejercicio de la llamada a subrutina de la Figura 17. El acercamiento a ver funcionando un microcontrolador para analizar esta secuencia es mucho menos detallada que el análisis ciclo a ciclo hecho anteriormente, pero se logra la misma meta (por ejemplo, muestra lo que pasa cuando la CPU ejecuta la secuencia). Después de estudiar el capítulo de programación, se debe intentar lo mismo con un programa más grande.

Se empieza el proceso preparando una hoja de trabajo como la mostrada en la Figura 18. Esta hoja incluye el programa con el código mnemónico y el código máquina. (Se podría escoger una alternativa utilizando un listado poniéndolo al lado de la hoja de trabajo). La hoja de trabajo también incluye los nombres de cada registro de la CPU en la parte superior de la hoja. Debajo hay un amplio cuadro para escribir nuevos valores de los registros cuando cambian en el curso del programa.

En esta hoja de trabajo, hay una área para guardar la traza de la pila (stack). Después se puede entender fácilmente cómo trabaja la pila, probablemente se puede dejar esta sección, pero es más instructivo dejarlo aquí por ahora.

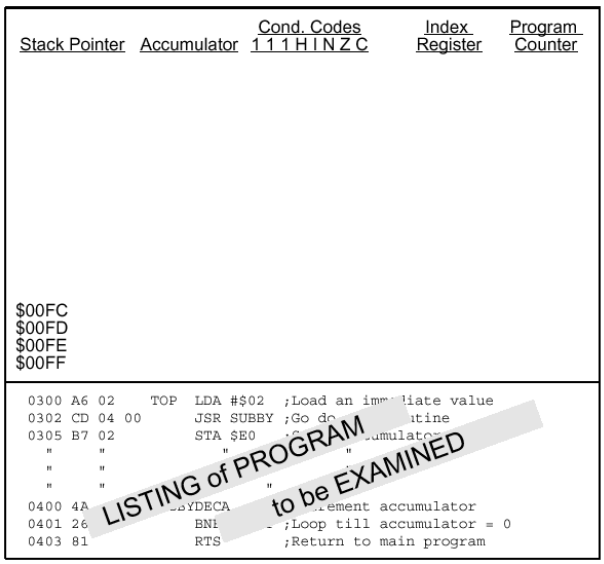


Figura 18. Hoja de trabajo para ver cómo funciona un microcontrolador.

Cuando un valor se guarda en la pila, se tacha el valor anterior y se escribe el nuevo valor directamente en una fila horizontal. También se debe actualizar el valor del SP (decrementado). Hay que tachar el valor anterior y escribir el nuevo valor bajo el registro SP que encabeza a la parte superior de la hoja de trabajo. Cuando un valor se recupera de la pila, se puede actualizar el valor del SP (incrementado), tachando el valor anterior y escribiendo el nuevo valor debajo de este. Entonces se puede leer el valor de la posición ahora apuntada por el SP y poner donde se quiera de la CPU (por ejemplo, en la mitad superior o inferior del PC).

La Figura 19 muestra cómo quedará la hoja de trabajo después de estar trabajando a través de la secuencia completa de JSR. Siguiendo los números entre paréntesis, se explica el proceso. Durante el proceso, se escriben muchos valores y después se tachan; se ha dibujado una línea desde cada número entre paréntesis hasta cada valor tachado para mostrar a qué número de referencia está aplicado.

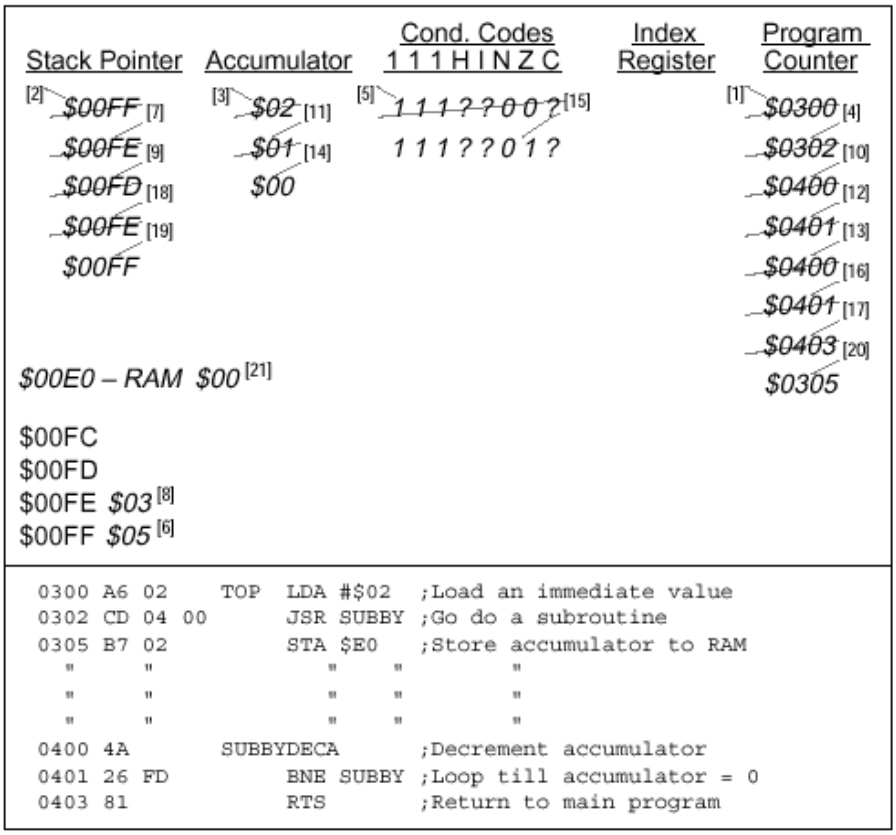


Figura 19. Hoja de trabajo completa

Empieza la secuencia:

- El PC debe estar apuntando a $0300 [1] y el SP debe estar apuntando a $00FF [2] (son valores aceptados de inicio).
- La CPU lee y ejecuta la instrucción LDA #$02 (carga el acumulador con el valor inmediato $02).
- Así que, se puede escribir $02 en la columna del acumulador [3] de la hoja de trabajo y reemplazar el valor del PC [4] con $0302, que es la dirección de la siguiente instrucción.
- La instrucción cargada en el acumulador afecta los bits N y Z del CCR. Puesto que el valor cargado fue $02, el bit Z y N se pondrán a 0 [5]. Esta información se puede encontrar en la sección **LDA** (Carga del Acumulador desde la Memoria) en el capítulo **Detalle del Juego de Instrucciones**.
- Puesto que los otros bits del CCR no están afectados por la instrucción LDA, no hay ninguna manera de saber como deben estar en este momento, por ahora los signos de interrogación se han puesto en las posiciones desconocidas [5].

A continuación:

- La CPU lee la instrucción JSR SUBBY. Temporalmente recuerda el valor $0305 que es la dirección donde la CPU debe regresar después de ejecutar la llamada a subrutina. La CPU guarda la parte baja de la dirección de retorno en la pila.
- Así que, se escribe $05 [6] en la posición apuntada por el SP ($00FF) y se decrementa el SP [7] a $00FE.
- La CPU guarda la parte alta de la dirección de retorno en la pila.

- Entonces se escribe $03 [8] a $00FE y otra vez se decrementa el SP [9], esta vez a $00FD.
- Para terminar con la instrucción JSR, se carga el PC con $0400 [10], que es la dirección de la llamada a subrutina.

- La CPU va a buscar la siguiente instrucción. Puesto que el PC es $0400, la CPU ejecuta la instrucción DECA, la primera instrucción de la subrutina.
- Se tacha $02 en la columna del acumulador y se escribe el nuevo valor $01 [11].
- También se cambia el PC a $0401 [12].
- Debido a que la instrucción DECA cambió el acumulador de $02 a $01 (qué no es cero o negativo), los bits Z y N permanecerán a 0. Cada vez que se modifica N y Z, se puede anotar el valor de ellos en la hoja de trabajo.
- La CPU ejecuta ahora la instrucción BNE SUBBY. Puesto que el bit Z está a 0, se reúne la condición de bifurcación y la CPU tomará la bifurcación. Se tacha $0401 bajo el PC y se escribe $0400 [13].
- La CPU ejecuta de nuevo la instrucción DECA. El acumulador se cambia ahora de $01 a $00 [14] (qué es 0 y no negativo); así, el bit Z está a 1 y el bit N permanece a 0 [15].
- El PC avanza a la siguiente instrucción [16].
- La CPU ejecuta ahora la instrucción BNE SUBBY, pero esta vez la condición de bifurcación no es verdadera (Z ahora se pone a 1) y no bifurcará. La CPU simplemente baja a la siguiente instrucción (la RTS a $0403).
- Actualiza el PC a $0403 [17].
- La instrucción RTS provoca que la CPU recupere el PC previo de la pila. Saca la parte alta del PC de la pila incrementando el SP a $00FE [18] y lee $03 de la posición $00FE.
- A continuación, saca la parte baja de la dirección de la pila, incrementando el SP a $00FF [19] y lee $05 de $00FF. La dirección recuperada de la pila reemplaza el valor del PC [20].
- Seguidamente la CPU lee la instrucción STA $E0 de la posición $0305. El flujo del programa devuelve el proceso al programa principal de donde salió cuando fue llamada la subrutina.
- La instrucción STA (modo de direccionamiento directo) escribe el valor del acumulador a la dirección directa $E0 ($00E0), que está en la RAM del MC68HC705J1A. Se puede ver, de la hoja de trabajo, que el valor actual en el acumulador es $00; por consiguiente, los ocho bits de esta posición de RAM estarán a 0. Puesto que la hoja de trabajo original no tenía una marca para grabar este valor en la RAM, se haría un espacio y se escribiría en él $00 [21].

Para un programa más grande, la hoja de trabajo tendrá muchos más valores tachados. La hoja de trabajo es un buen ejercicio de aprendizaje, pero como que un programador gana con la experiencia, el proceso se simplificará. En el capítulo de programación, se verá una herramienta de desarrollo llamada **simulador**, que automatiza el proceso de desarrollo del microcontrolador. El simulador es un programa para el microcontrolador en el que corre desde un ordenador personal. Los contenidos actuales de los registros y las posiciones de memoria se muestran en la pantalla ordenador personal.

Una de las primeras simplificaciones que se podría hacer en una hoja de trabajo, sería dejar de guardar los valores del PC porque se aprende a confiar a la CPU que ya se cuida de esto. Otra simplificación, es dejar de guarda la pista de los códigos de condición. Cuando se encuentra una instrucción 'branch' de la que depende un bit del código de condición, se puede trabajar mentalmente hacia atrás para decidir si debe tomarse o no la bifurcación.

A continuación, se saltaría el almacenamiento de los valores de la pila, aunque es una buena idea para guardar la pista del propio valor del SP. Es bastante común tener errores de programación, que son el resultado de valores incorrectos en el SP. Un principio fundamental de trabajo de la pila es que sobre un periodo de tiempo, se debe poner en la pila el mismo número de datos como el que debe quitarse. Tal como que se deben emparejar los paréntesis izquierdos con los paréntesis derechos en una fórmula matemática, también se debe hacer en un programa. Las instrucciones JSRs y BSRs deben ser emparejadas para una subsiguiente RTS. Los errores que causan el incumplimiento de estas reglas, aparecerán como valores erróneos del SP, mientras funciona el microcontrolador.

Incluso un programador experimentado que programa ocasionalmente un microcontrolador para resolver algún problema difícil, el procedimiento sería usar un modo menos formal que lo que se explica aquí, pero todavía aumenta ponerse en el papel de la CPU y trabajar tal como pasa cuando se ejecuta el programa.

Reset

El 'reset' (vuelta al estado inicial) se usa para forzar al microcontrolador arrancar en un lugar conocido (en una dirección determinada). También los periféricos, así como muchos bits de estado y de control, son forzados a restablecer su estado inicial como resultado de un 'reset'.

Las siguientes acciones internas ocurren como resultado de cualquier 'reset' en el microcontrolador:

1. Todos los registros de dirección de datos se ponen a 0 (entrada).
2. El indicador de pila (SP) se fuerza a $00FF.
3. El bit I del registro CCR se pone a 1 para inhibir las interrupciones enmascarables.
4. El indicador de Interrupción Externa es borrado.
5. El indicador de STOP es borrado.
6. El indicador de WAIT es borrado.

Cuando un sistema microcontrolador desactiva el 'reset', el contador de programa se carga de las dos posiciones de memoria más altas ($07FE y $07FF en un MC68HC705J1A). El valor $07FE se carga en la parte alta del byte del PC y el valor $07FF se carga en el byte de la parte baja del PC. A esto se le llama "sacar el vector de reset". En este punto, la CPU empieza a cargar y ejecutar las instrucciones, empezando en la dirección que fue guardado en el vector de 'reset'.

Cualquiera de estas condiciones puede provocar un 'reset' en la MCU del MC68HC705J1A:
1. Externamente, una señal de entrada en estado bajo, en el terminal RESET.
2. Con el 'Power on reset' interno (POR).
3. Por desbordamiento en el temporizador del 'WatchDog' interno (COP).
4. Por un intento de ejecutar una instrucción desde una dirección ilegal.

Pin de $\overline{\text{RESET}}$

A este pin se puede conectar un interruptor externo o un circuito, para permitir al sistema un 'reset' manual.

Power-On Reset

El 'power-on reset' ocurre, cuando una transición positiva se detecta en VDD. El 'Power-On Reset' se usa estrictamente para condiciones de desconexiones de alimentación y no se debe usar para detectar cualquier perdida en la tensión de alimentación. Para esto, se debe utilizar un circuito de inhibición por bajo voltaje (LVI), para detectar cualquier pérdida de tensión de alimentación.

La circuitería de 'power-on' mantiene un retardo de 4064 ciclos de tiempo para que el oscilador se ponga activo. Si el pin externo de $\overline{\text{RESET}}$ está a un nivel bajo, al final de los 4064 ciclos de retardo, el procesador permanece en la condición de 'reset' hasta que el $\overline{\text{RESET}}$ se ponga en estado alto.

Reset por Watchdog Timer

El microcontrolador con un "control de proceso correcto" (COP) es el sistema WatchDog Timer, está pensado para descubrir errores de software. Cuando el COP se está usando, el software es responsable para guardar un cronómetro llamado 'watchdog' que forma autónoma. Si el cronómetro del 'watch dog' se para, es una indicación que el software ya no está ejecutándose en la secuencia deseada; así que se inicia un 'reset' en el sistema.

Para habilitar o deshabilitar el 'reset' (COP) se puede usar un bit de control en el registro de control de configuración (máscara no-volátil). Si se habilita el COP, el programa de trabajo debe escribir un 0 periódicamente en el bit del COPC en el registro de control COPR. Para información sobre los tiempos de interrupción del COP se puede ver en las hojas de datos técnicos del MC68HC705J1A (referencia MC68HC705J1A/D). Algunos miembros de la familia MC68HC05 tienen diferentes sistemas COP 'watchdog timer'.

Reset por una Dirección ilegal

Si un programa se escribe incorrectamente, es posible que la CPU intente saltar o bifurcar a una dirección que no tiene en memoria. Si pasa esto, la CPU continuará leyendo datos (aunque serán valores imprevisibles) e intentará actuar con estos datos como si fueran de un programa. Éstas instrucciones sin sentido, pueden causar a la CPU escribir datos inesperados a la memoria o en el registro de direcciones. Esta situación se llama huida del programa.

Para impedir esta condición de huida, hay que detectar una dirección ilegal en el circuito MC68HC705J1A. Si la CPU intenta sacar una instrucción de una dirección que no está en la EPROM ($0300–$07CF, $07F0–$07FF), ROM ($07EE–$07EF) o RAM ($00C0–$00FF), se genera un 'reset' para obligar volver a empezar al programa.

Interrupciones

A veces es útil interrumpir un proceso normal para responder a algunos eventos inusuales. Por ejemplo, el MC68HC705J1A pueden interrumpirse por cualquiera de estas fuentes de interrupción:

1. Aplicando un 0 en el pin de interrupción externa $\overline{\text{IRQ}}$.
2. Aplicando un 1 en cualquiera de los pins PA3–PA0, con tal de que la función de interrupción del puerto esté habilitada.
3. Por un desbordamiento (overflow) del temporizador (timer) (TOF) o una interrupción en tiempo real (RTIF) solicitada desde el temporizador multifuncional interno del sistema, si éste está habilitado.
4. Por la instrucción de interrupción por software (SWI).

Si se hace una interrupción mientras la CPU está ejecutando una instrucción, la instrucción se completa antes de que la CPU responda a la interrupción.

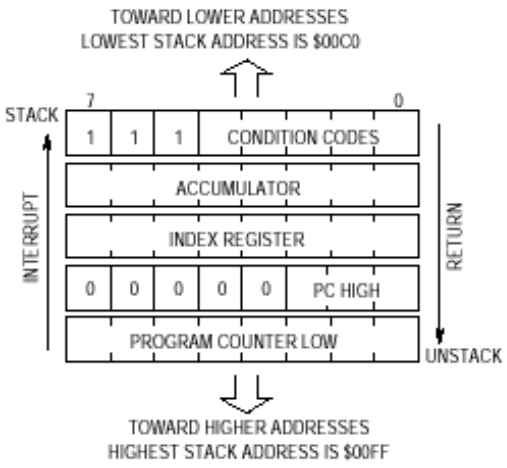
Las interrupciones se pueden inhibir poniendo el bit I del registro de código de condición (CCR) o poniendo a 0 los bits de control de la interrupción individual habilitada para cada fuente de interrupción. El 'reset' fuerza al bit I a 1 y pone a 0 todos los bits de interrupciones locales habilitados, para impedir interrupciones durante la inicialización del procedimiento. Cuando el bit I es 1, no se reconoce ninguna interrupción (excepto la instrucción SWI). Sin embargo, las fuentes de interrupción puede todavía registrar una demanda, que será respetada un momento más tarde, cuando el bit I se ponga a 0.

La Figura 21 muestra cómo las interrupciones encajan en el flujo normal de las instrucciones en la CPU. Las interrupciones causan que los registros del procesador se guarden en la pila y el bit de interrupción enmascarada sea puesto a 1, para prevenir interrupciones adicionales, hasta que la interrupción presente haya sido terminada. El vector de interrupción apropiado apunta entonces a los puntos de la dirección de arranque de la rutina de servicio de interrupción (Tabla 10).

En la realización de la rutina de servicio de interrupción, una instrucción RTI (qué normalmente es la última instrucción de un servicio de rutina de interrupción) causa que se recuperen los contenidos del registro, de la pila. Puesto que el contador de programa está cargado con el valor que fue previamente guardado en la pila y el proceso continúa desde donde salió antes de la interrupción. La Figura 20 muestra los registros que son restablecidos de la pila en el orden opuesto que fueron guardados.

Fuente de Interrupción o Reset	Dirección del Vector
Timer Interno	$07F8, $07F9
$\overline{\text{IRQ}}$ o Pins del puerto A	$07FA, $07FB
Intrucción SWI	$07FC, $07FD
Reset (POR, LVI, Pin, COP, o dirección ilegal	$07FE, $07FF

Tabla 10. Vectores de Dirección para Interrupciones y Reset en el MC68HC705J1A



Nota: Cuando ocurre una interrupción, los registros de la CPU son guardados en la pila (stack) en este orden: PCL, PCH, X, A, CCR. Al volver de una interrupción, los registros se recuperan de la pila (stack) en el orden inverso.

Figura 20. Orden de guardar los registros en la pila cuando se produce una Interrupción

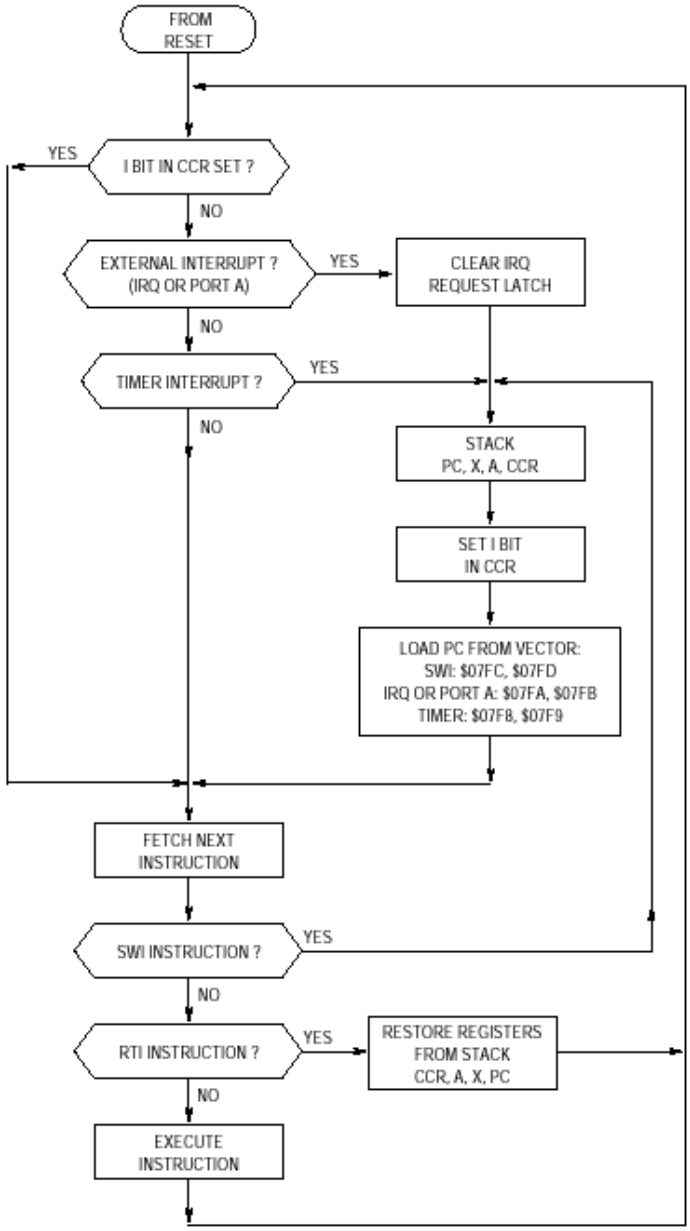


Figura 21. Diagrama de Flujo de las Interrupciones Hardware

Interrupciones externas

Las interrupciones externas vienen del pin $\overline{IRQ}$ o de los bits 3–0 del puerto A, si este puerto está configurado para las interrupciones. En el MC68HC705J1A, la sensibilidad del pin $\overline{IRQ}$ es programable por software.

En el MC68HC705J1A, están disponibles dos opciones de interrupciones externas:

- Activada por flanco.
- Por flanco negativo y nivel de disparo

El MC68HC705J1A usa un bit en la opción de registro en la posición $07F1 para configurar la sensibilidad del pin $\overline{IRQ}$. El pin $\overline{IRQ}$ es verdadero con el nivel bajo y las interrupciones en el puerto A son verdaderas con el nivel alto.

Cuando se reconoce una interrupción, el estado actual de la CPU se envía hacia la pila (stack) y el bit I se pone a 1. Esto enmascara interrupciones adicionales hasta la presente que está servida. La dirección de la interrupción externa a la rutina de servicio, está especificada por los contenidos de las posiciones de memoria $07FA y $07FB.

Interrupciones de Periféricos internos

Los microcontroladores incluyen a menudo periféricos internos que pueden generar interrupciones a la CPU. El 'timer' en el MC68HC705J1A es un ejemplo de periférico. Las interrupciones de los periféricos internos simplemente trabajan como las interrupciones externas, sólo que normalmente tienen vectores de interrupción separados para cada sistema periférico interno.

Interrupción por Software (SWI)

La interrupción por software es una instrucción ejecutable. La acción de la instrucción SWI es similar a las interrupciones por hardware. Una instrucción SWI se ejecuta sin tener en cuenta el estado de la máscara de interrupción (bit I) en el registro de código de condición. La dirección del servicio de interrupción de la rutina, se especifica por los contenidos de la posición de memoria $07FC y $07FD en un MC68HC705J1A.

Latencia de Interrupción

Aunque se piensa de las interrupciones causan la inmediata parada normal del proceso de la CPU para responder a la demanda de una interrupción, esto realmente no es así. Hay un pequeño retardo entre que se solicita una interrupción hasta que la CPU realmente puede responder. Primero, la CPU debe terminar cualquier instrucción que esté en marcha en el momento de la petición de interrupción. (La CPU no sabría continuar el proceso después de manejar la interrupción, si se hubiera detenido en el medio de una instrucción). Segundo, la CPU debe hacer un registro de lo que estaba haciendo antes de que respondiera a la interrupción. La CPU hace esto, guardando una copia de los contenidos de todos sus registros, incluso el contador de programa, en la pila (stack). Después de que la interrupción se ha servido, la CPU recupera esta información en orden inverso y continua con el proceso normal.

La latencia de la interrupción es el número total de ciclos de la CPU (tiempo) desde el inicio de la demanda de interrupción hasta que la CPU empieza a ejecutar la primera instrucción de la rutina de servicio de interrupción. Este retraso depende en si o no, la máscara de interrupción se pone a 1 cuando se solicita la interrupción. Si el bit I se pone a 1, el retraso podría ser indefinido y podría depender en si de una instrucción de puesta a 0 del bit I, para que la interrupción pueda ser reconocida por la CPU. En el caso normal, donde el bit I está a 0 cuando se solicita la interrupción, la latencia consistirá en acabar la instrucción actual y guardará los registros en la pila y cargará el vector de interrupción (dirección de la rutina de servicio de interrupción) en el contador de programa.

La instrucción más larga (tiempo de ejecución) en el MC68HC05 es la instrucción multiplicación (MUL) que necesita 11 ciclos. Si la CPU justo en el momento de empezar a ejecutar una instrucción MUL, se solicita una interrupción, se produce un retraso de 11 ciclos antes de que pudiera responder la CPU. La CPU necesita nueve ciclos para guardar una copia de sus registros en la pila y para sacar el vector de interrupción. La latencia total en el peor de los casos, si el bit I está a 0 y justo empezar una instrucción MUL, será de 20 ciclos (11 + 9).

El bit I se pone a 1 cuando la CPU responde a una interrupción, para que (normalmente) una nueva interrupción no sea reconocida, hasta que la actual haya sido manejada. En un sistema que tiene más de una fuente de interrupción, debe ser calculado el tiempo de la ejecución para la rutina de servicio de interrupción más larga, para determinar la latencia de interrupción, en el peor de los casos, para otras fuentes de interrupción.

Interrupciones anidadas

En raros casos, una rutina de servicio de interrupción puede ser tan larga a ejecutar que la latencia, en el peor de los casos, para otras interrupciones en el sistema sea demasiado tiempo. En semejante caso, las instrucciones en la larga rutina de servicio de interrupción se puede poner el bit I a 0 y permitir así una nueva interrupción para ser reconocida antes de que la primera rutina de servicio de interrupción se termine. Si se solicita una nueva interrupción mientras la CPU ya está manejando una interrupción, este tipo de interrupción se llama anidada. Se debe tener mucho cuidado si se permite una interrupción anidada porque la pila debe tener bastante espacio para soportar más de una copia de los registros de la CPU. En los microcontroladores pequeños como en el MC68HC05K1, la pila es pequeña y no se recomiendan las interrupciones anidadas.

Resumen

En la arquitectura MC68HC05, se conectan cinco registros directamente dentro de la CPU y no forman parte del mapa de memoria. Toda la otra información disponible en la CPU se localiza en una serie de posiciones de memoria de 8-bits. El **mapa de memoria** muestra los nombres y los tipos de memoria de todas las posiciones que son accesibles a la CPU. La expresión mapeado de memoria de E/S significa que la CPU trata las I/O y los registros de control como cualquiera tipo de memoria. (Algunas arquitecturas de microcontrolador separan los registros de E/S del espacio de memoria de programa y usa instrucciones separadas para acceder a las posiciones de E/S.)

Para empezarse en un lugar conocido, en un microcontrolador debe haber '**reset**'. El 'reset' fuerza a los sistemas periféricos internos y las E/S, a las condiciones conocidas y carga el contador de programa con una dirección de arranque conocida. El usuario especifica la posición de inicio deseada, poniendo los bytes de la parte superior e inferior de esta dirección en las posiciones del vector de reset ($07FE y $07FF en el MC68HC705J1A).

La CPU usa el registro indicador de la **pila o stack pointer** (SP) para implementar una pila de último en entrar primero en salir en la memoria RAM. Esta pila mantiene las direcciones de retorno mientras que la CPU está ejecutando una subrutina y mantiene los contenidos anteriores de todos los registros de la CPU mientras que la CPU está ejecutando una secuencia de interrupción. Recuperando esta información de la pila, la CPU puede continuar de donde se fue antes de empezar la subrutina o la interrupción.

Los microcontroladores usan un reloj de alta velocidad para pasar a través cada pequeño paso de cada operación. Aunque cada instrucción necesita varios ciclos de este reloj, es tan rápido que las operaciones parecen instantáneas a un humano. Un MC68HC705J1A pueden ejecutar aproximadamente 500,000 instrucciones por segundo.

Una CPU ve un programa como una secuencia lineal de números binarios de 8-bits. Los **'opcodes'** de cada instrucción y los datos se mezclan en esta sucesión pero la CPU reconoce los límites de la instrucción porque cada 'opcode' dice a la CPU cuántos bytes de datos de **operando** van con cada 'opcode' de la instrucción.

Ver funcionando un microcontrolador, es un ejercicio de aprendizaje donde se pretende ver como una CPU está ejecutando un programa.

Un 'reset' puede ser provocado por condiciones internas o externas. Un 'reset' por el pin de un puerto permite por una causa externa comenzar hacer un 'reset'. Un 'watchdog timer' y una dirección ilegal detectada por el sistema pueden provocar un 'reset' en el software para no ejecutar una secuencia propuesta. Las interrupciones provocan el paro del proceso, en la CPU, del programa principal temporalmente para responder a la interrupción. Todos los registros de la CPU se guardan en la pila para que la CPU pueda regresar de donde salió del programa principal en cuanto la interrupción sea manejada.

Las interrupciones se pueden inhibir globalmente poniendo el bit I en el CCR o localmente poniendo a 0 los bits de control para cada fuente de interrupción. Las peticiones todavía se pueden registrar mientras se inhiben las interrupciones para que la CPU pueda responder en cuanto las interrupciones se rehabiliten. SWI es una instrucción y no puede inhibirse.

La latencia de la interrupción es el retraso desde que se solicita una interrupción hasta que la CPU empieza a ejecutar la primera instrucción en el programa de respuesta de interrupción. Cuando una CPU responde a una nueva interrupción mientras que ya se está procesando una interrupción (qué normalmente no se permite), se llama interrupción anidada.

Juego de Intrucciones del MC68HC05

Índice

Introducción

El juego de instrucciones de un microcontrolador es su vocabulario. Este capítulo describe la CPU y el conjunto de instrucciones del MC68HC05. En el capítulo **Detalles del Juego de Instrucciones** contiene las descripciones detalladas de cada instrucción del MC68HC05 y se puede usar como una referencia. Se describen las mismas instrucciones en grupos de trabajo funcionalmente similares. También se describe la estructura y los modos de direccionamiento. Los modos de direccionamiento se refieren a las varias maneras de que una CPU puede acceder a los operandos para una instrucción.

Unidad Central de Proceso (CPU)

La CPU del MC68HC05 es responsable de ejecutar todas las instrucciones de software en su secuencia programada para una aplicación específica. La Figura 22 muestra el diagrama de bloques de la CPU MC68HC05.

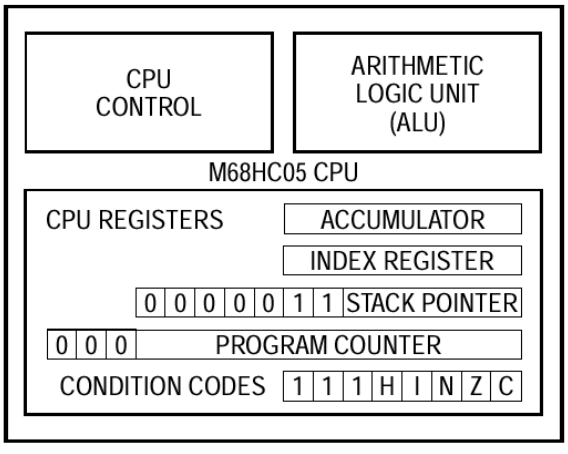


Figura 22. Diagrama de bloques de la CPU MC68HC05

Unidad Aritmética/Lógica (ALU)

La unidad aritmética lógica **(ALU)** se usa para realizar la aritmética y las operaciones lógicas definidas por el juego de instrucciones.

Los varios circuitos de funcionamientos de aritmética binaria decodifican la instrucción en el registro de la instrucción y preparó la ALU para la función deseada. La mayoría de la aritmética binaria se basa en el algoritmo suma y substracción que se llevada a cabo como una suma negativa. La multiplicación no se realiza como una instrucción discreta, pero sí como una cadena de operaciones suma y desplazamiento dentro de la ALU bajo el control de la lógica de control de la CPU. La instrucción multiplicación (MUL) requiere 11 ciclos de proceso interno para completar esta cadena de operaciones.

Control de la CPU

La circuitería de control de la CPU gestiona los elementos lógicos de la ALU para llevar a cabo las operaciones requeridas. Un elemento central de control de la CPU es el **decodificador de instrucciones**. Cada 'opcode' se decodifica para determinar cuántos operandos se necesitan y qué sucesión de pasos se requieren para completar la instrucción. Cuando se ha terminado una instrucción, se lee el siguiente 'opcode' y se decodifica.

Registros de la CPU

La CPU contiene cinco registros, mostrados en la Figura 23. Los registros en la CPU son como memorias dentro del microprocesador (pero, no forman parte del mapa de memoria). El conjunto de registros en una CPU a veces se llama **modelo de programación**. Un programador experimentado puede decir mucho sobre la forma del modelo de la programación de un microcontrolador.

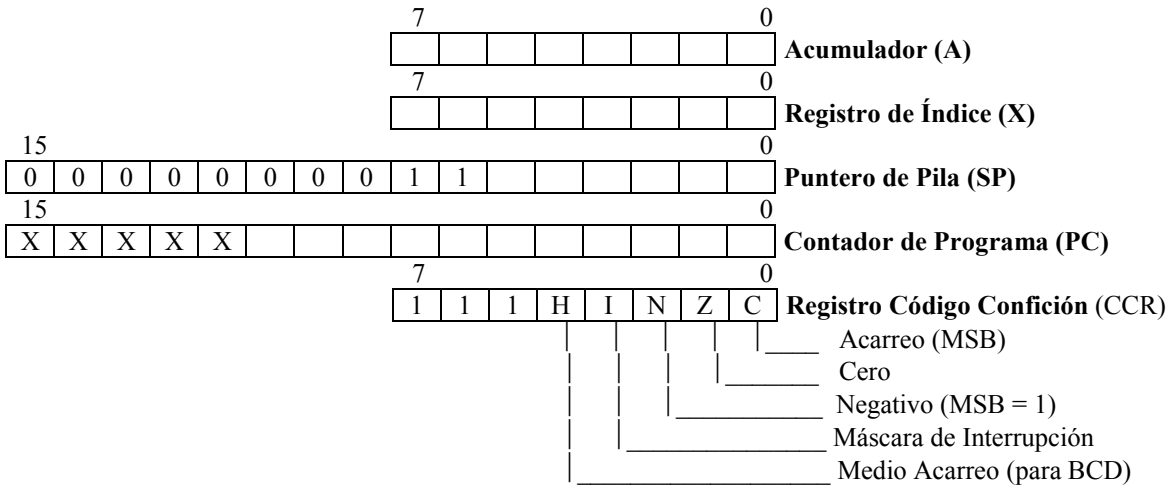


Figura 23. Modelo de Programación

(A) Acumulador

El acumulador es un registro de 8-bits de propósito general utilizado para memorizar los operandos, resultados de los cálculos aritméticos y las manipulaciones de datos. También se puede acceder directamente a la CPU para las operaciones no aritméticas. El acumulador se usa durante la ejecución de un programa, cuando el contenido de alguna posición de memoria se carga en el acumulador. También, al almacenar una instrucción causa que los contenidos del acumulador sean guardados en algunas posiciones de memoria determinadas.

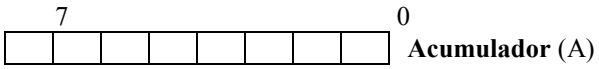


Figura 24. Acumulador (A)

(X) Registro de índice

El registro del índice se usa para los modos de direccionamiento indexado o puede ser usado como un acumulador auxiliar. Este registro de 8-bits puede ser cargado directamente con cualquier cosa o desde la memoria, sus contenidos pueden guardarse en memoria o se pueden comparar con los contenidos de la memoria.

En instrucciones indexadas, el registro X proporciona un valor de 8-bits que se añade al valor de una instrucción dada para crear una dirección efectiva. El valor de la instrucción dada puede tener una longitud de 0, 1 o 2 bytes.

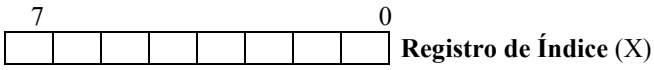


Figura 25. Registro de índice (X)

(CCR) Registro de Código Condición

El registro de código de condición contiene una máscara de interrupción y cuatro indicadores de estado que reflejan los resultados de la aritmética y otras operaciones de la CPU. Los cinco indicadores son:

- Medio acarreo (H) , para operaciones BCD de 8 bits.
- Negativo (N)
- Cero (Z)
- Desbordamiento (V)
- Acarreo/Acarreo en substracción (C)

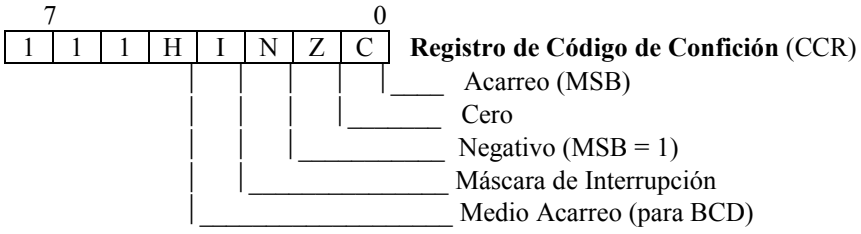


Figura 26. Registro de Código de condición (CCR)

(H) Bit Medio acarreo

El indicador de medio acarreo se usa para operaciones aritméticas en BCD y está afectado por las instrucciones suma ADD o ADC. El bit H se pone a un 1 cuando ocurre un acarreo de la parte baja de un dígito hexadecimal en los bits 3–0 y en la parte alta del dígito en los bits 7–4. Después de una suma binaria de dos dígitos con valor BCD, este bit de medio acarreo es un bit de la información necesaria para restaurar el resultado a un valor BCD válido.

(I) Bit Máscara de interrupción

El **bit I** no es un indicador de estado, pero es un bit de máscara de interrupción que desactiva todas las fuentes de interrupción enmascarables cuando el bit I se pone a 1. Se habilitan interrupciones cuando el bit I está a 0. Cuando ocurre cualquier interrupción, el bit I se pone a 1 automáticamente después de que los registros son apilados, pero antes de sacar el vector de interrupción.

Si ocurre una interrupción externa mientras el bit I está en 1, la interrupción es enclavada y procesada después de poner a 0 el bit I; por consiguiente, no se pierde ninguna interrupción del pin $\overline{\text{IRQ}}$ debido al bit I continua estando en 1.

Después de que se ha procesado una interrupción, una instrucción de retorno de interrupción (RTI) provoca que los registros serán restablecidos a sus valores anteriores. Normalmente, el bit I estará a 0 después de ejecutarse una RTI. Después de cualquiera 'reset', el bit I se pone a 1 y sólo se pondrá a 0 por una instrucción software.

(N) Bit negativo

El **bit N** se pone a 1 cuando el resultado de la última manipulación aritmética, lógica o de datos es negativo. Los valores complemento a dos son considerados negativos si el bit más significativo es un 1. El bit N tiene otros usos, como la indicación del MSB de un registro o de una posición de memoria. Para analizar este bit, hay que cargar simplemente el acumulador con el contenido de esta posición.

(Z) Bit Cero

El **bit Z** se pone a 1 cuando el resultado de la última manipulación aritmética, lógica o de datos es 0. Una instrucción de comparación substrae un valor de la posición de memoria que se prueba. Si los valores fueran iguales antes de la comparación, el bit Z se pondrá a 1.

(C) Bit Carry/Borrow

El **bit C** se usa para indicar si hay un 'carry' (acarreo en una suma) o 'borrow' (acarreo como resultado de una substracción). Desplaza y opera las instrucciones de rotación con y a través del bit de acarreo para facilitar operaciones de desplazamiento de múltiples palabras. Este bit es también afectado durante el bit de prueba y instrucciones 'branch'.

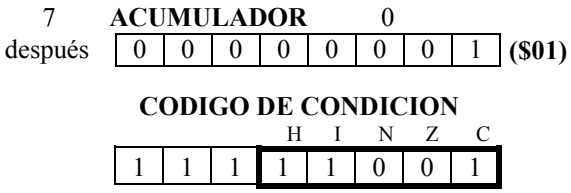
La Figura 27 es un ejemplo para ver la manera de que el bit de código de condición se ve afectado por las operaciones aritméticas. El bit H no es útil después de esta operación, porque el acumulador no fue un valor BCD válido antes de la operación.

Asumiendo valores iniciales en el acumulador y los códigos de condición:

Se ejecuta esta instrucción:

```
- - - - -  AB 02          ADD    #2          ;ADD 2 al Acumulador
```

Los códigos de condición y el acumulador reflejan los resultados de la instrucción ADD:



H– Se pone a 1 porque ha habido un acarreo del bit 3 al bit 4 del acumulador
I – No cambia
N– Se pone a 0 porque el resultado no es negativo (el bit 7 del acumulador es 0)
Z– Se pone a 0 porque el resultado no es 0
C– Se pone a 1 porque ha habido un acarreo del bit 7 del acumulador

Figura 27. Cómo los códigos de condición son afectados por las operaciones aritméticas

Contador de Programa

El contador de programa es un registro de 16-bits que contiene la dirección de la siguiente instrucción o el operando de la instrucción sacada por el procesador. En la mayoría de variantes del MC68HC05, algunos bits de la parte alta del contador de programa no se usan y siempre están a 0. El MC68HC705J1A utiliza sólo 11 bits del contador de programa, con los cinco bits de la parte alta siempre a 0. El número de bits útiles en el contador de programa se apareja exactamente con el número de líneas de dirección implementadas en el microcontrolador.

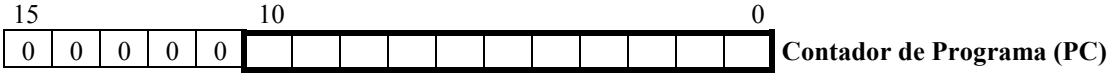


Figura 28. Contador de Programa (PC)

Normalmente, el contador de programa avanza una posición de memoria en por instrucción, al tiempo que esta es leída. Las operaciones como salto (jump), bifurcación (branch) e interrupciones provocan la carga del contador de programa con otra dirección de memoria diferente, que la siguiente posición secuencial.

Puntero de Pila (Stack Pointer)

El puntero de pila debe tener tantos bits como líneas de dirección; en el MC68HC705J1A significa que el SP es un registro de 11-bits. Durante un 'reset' o una instrucción 'reset' del puntero de pila (RSP), el puntero de pila se pone a 1 en la posición $00FF. Entonces, el puntero de pila se decrementa tanto como datos enviados hacia la pila y es incrementado tanto como datos son sacados de la pila.

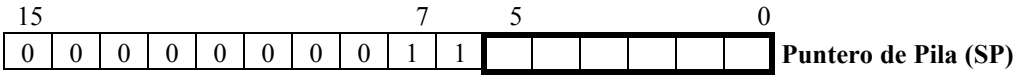


Figura 29. Indicador de Pila (SP)

Algunos modelos del MC68HC05 permiten a la pila usar hasta 64 posiciones ($00FF a $00C0), pero las versiones más pequeñas permiten sólo 32 bytes de pila ($00FF a $00E0). En el MC68HC705J1A, los cinco bits más significativos (MSB) del SP son permanentemente puestos a 00011. Estos cinco bits se añaden a los seis bits menos significativos para producir una dirección dentro del rango de $00FF a $00C0. Las subrutinas e interrupciones pueden usar hasta 64 posiciones (en decimal). Si se excede de las 64 posiciones, el puntero de pila sobreescribe la información previamente guardada en otros registros. Una llamada a subrutina, ocupa dos posiciones en la pila; una interrupción usa cinco posiciones.

Modos de Direccionamiento

La potencia de cualquier microcontrolador está en su habilidad de acceder a la memoria. Los **modos de direccionamiento** de la CPU proporcionan esta capacidad. Los modos de direccionamiento definen la manera en la que una instrucción obtendrá los datos requeridos para su ejecución. Debido a los diferentes modos de direccionamiento, una instrucción puede acceder al operando de seis maneras diferentes. De esta manera, los modos de direccionamiento extienden las 62 instrucciones básicas de la familia MC68HC05 a 210 'opcodes' distintos.

Los modos de direccionamiento usados, referente a la memoria son:

- *Inherente*
- *Inmediato*
- *Extendido*
- *Directo*
- *Indexado sin desplazamiento, con desplazamientos de 8-bits y con desplazamientos de 16-bits*
- *Relativo*

Las instrucciones inherentes no necesitan acceder a memoria, son aquellas instrucciones de un solo byte. En el MC68HC05 más pequeño, toda la RAM y los registros de E/S están dentro del área de memoria $0000–$00FF para que puedan usarse los dos bytes de las instrucciones en el modo de direccionamiento directo. El direccionamiento extendido usa instrucciones de 3-bytes para alcanzar datos en cualquier parte del espacio de memoria. Los varios modos de direccionamiento hacen posible localizar tablas de datos, código, tablas de conversión y tablas escalares en cualquier parte del espacio de memoria. Accesos cortos indexados son instrucciones de un solo byte, pero la instrucción más larga (tres bytes) permite acceder a tablas en cualquier parte de la memoria.

En los párrafos siguientes se proporciona una descripción general con ejemplos de los varios modos de direccionamiento. El término **dirección efectiva** (EA) se usa para indicar la dirección de memoria de donde se ha leído o guardado el argumento para una instrucción. Para más detalles de los modos de direccionamiento y para una descripción de cada instrucción, está disponible en el capítulo titulado **Detalles del Juego de Instrucciones**.

La información proporcionada en los ejemplos de programa en lenguaje ensamblador, se usan varios símbolos para identificar los varios tipos de números en los que se presentan en un programa. Estos símbolos son:

1. Un **espacio en blanco** o ningún símbolo, indica un número decimal.
2. Un símbolo **$** precedido de un número, indica que es un número hexadecimal; por ejemplo, $24 es 24 en hexadecimal o el equivalente en decimal que es 36.
3. Un símbolo **#**, indica un operando inmediato y el número se encuentra en la posición que sigue al 'opcode'. Una variedad de símbolos y expresiones se pueden usar inmediatamente después del carácter #. Puesto que no

todos los ensambladores usan la misma sintaxis y caracteres especiales, hay que referirse a la documentación de cada ensamblador particular, que se use.

Prefijo	Indica que el valor que sigue es. . .
Nada	Decimal
$	Hexadecimal
@	Octal
%	Binario
‘	Simple carácter ASCII

Para cada modo de direccionamiento, se explica en detalle una instrucción como ejemplo. Estas explicaciones describen lo que pasa en la CPU durante cada ciclo de reloj del proceso de la instrucción. Los números entre corchetes **[ ]**,se refieren a un ciclo de reloj específico de la CPU.

Modo de Direccionamiento Inherente

En el modo de direccionamiento inherente, toda la información requerida para la operación está ya inherentemente conocida por la CPU y no se necesita ningún operando externo desde la memoria o desde el programa. Los operandos, si hay alguno, sólo son el registro de índice y el acumulador; y siempre son instrucciones de 1 byte.

Listado de un Programa Ejemplo:

```
0300    4C      INCA                    ;Incrementa el acumulador
```

Secuencia de ejecución:
$0300 $4C [1], [2], [3]

Explicación:
[1] La CPU lee el ‘opcode’ $4C, incrementa el acumulador.
[2] y [3] La CPU lee el valor del acumulador, añade uno a él, guarda el nuevo valor en el acumulador y ajusta el bit del indicador de código de condición, como necesario.

A continuación se muestra una lista de todas las instrucciones del MC68HC05 que se pueden usar con el modo de direccionamiento inherente.

Instrucción	Código mnemotécnico
Desplazamiento aritmético izquierdo	ASLA,ASLX
Desplazamiento aritmético derecho	ASRA, ASRX
Pone a 0 el bit de Acarreo	CLC
Pone a 0 el bit de Máscara de interrupción	CLI
Pone a 0	CLRA, CLRX
Complementa	COMA, COMX
Decrementa	DECA, DECX
Incrementa	INCA, INCX
Desplazamiento lógico a la izquierda	LSLA, LSLX
Desplazamiento lógico a la derecha	LSRA, LSRX
Multiplicación	MUL
Negación	NEGA, NEGX
Ninguna operación	NOP
Rotación a la izquierda con Acarreo	ROLA, ROLX
Rotación a la derecha con Acarreo	RORA, RORX
Reset del Puntero de Pila	RSP
Retorno de una Interrupción	RTI
Retorno de una Subrutina	RTS
Pone a 1 el Bit de Acarreo	SEC
Pone a 1 el Bit de Máscara de Interrupción	SEI
Habilita IRQ, para el oscilador	STOP

Interrupción por Software	SWI
Transfiere al acumulador el registro de índice	TAX
Prueba para negativo o cero	TSTA, TSTX
Transfiere el registro de índice al acumulador	TXA
Habilita Interrupción, para el procesador	WAIT

Modo de Direccionamiento Inmediato

En el modo de direccionamiento inmediato, el operando está contenido en el byte siguiente al 'opcode' inmediato. Este modo se usa para guardar un valor o una constante que es conocida en el momento que el programa se escribe y qué no se cambia durante la ejecución del programa. Éstas son instrucciones de 2-bytes, uno para los 'opcode' y otro para los datos inmediatos.

Listado de Programa Ejemplo:

```
0300    A6 03   LDA    #$03     ;Carga el acumulador  con el valor inmediatamente siguiente
```

Secuencia de ejecución:
$0300 $A6 [1]
$0301 $03 [2]

Explicación:
[1] La CPU lee el 'opcode' $A6, carga el acumulador con el valor inmediatamente seguido al 'opcode'.
[2] Entonces la CPU lee los datos $03 inmediatos de la posición $0301 y carga $03 en el acumulador.

A continuación se muestra una lista de todas las instrucciones MC68HC05 que pueden usar el modo de direccionamiento inmediato.

Instrucción	Código mnemotécnico
Suma con acarreo	ADC
Suma	ADD
AND lógico	AND
Bit de prueba de memoria con el acumulador	BIT
Compara el acumulador con la memoria	CMP
Compara el registro de índice con la memoria	CPX
OR-Exclusiva de la memoria con el acumulador	EOR
Carga el acumulador desde la memoria	LIDA
Carga el registro de índice desde la memoria	LDX
OR-Inclusiva	ORA
Substracción con Acarreo	SBC
Substracción	SUB

Modo de Direccionamiento Extendido

En el modo de direccionamiento extendido, la dirección del operando está contenida en los dos bytes que siguen al 'opcode'. El direccionamiento extendido hace referencia a cualquier posición en el espacio de memoria de la MCU incluyendo E/S, RAM, ROM y EPROM. Las instrucciones del modo de direccionamiento extendido son tres bytes, uno para los 'opcode' y dos para la dirección del operando.

Listado de Programa Ejemplo:

```
0300    C6 06 E5        LDA $06E5       ;Carga el acumulador de la dirección extendida
```

Secuencia de la ejecución:
$0300 $C6 [1]
$0301 $06 [2]
$0302 $E5 [3] y [4]

Explicación:
[1] La CPU lee el 'opcode' $C6, carga el acumulador usando el modo de direccionamiento extendido.
[2] Entonces la CPU lee $06 desde la posición $0301. Este $06 es interpretado como la parte alta de la mitad de una dirección.
[3] La CPU lee $E5 de la posición $0302. Este $E5 es interpretado como la parte baja de la mitad de una dirección.
[4] La CPU añade $06 internamente al $E5 leído para formar la dirección completa ($06E5). La CPU lee cualquier valor contenido en la posición $06E5 en el acumulador.

A continuación se muestra un listado de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento extendido.

Instrucción	Código mnemotécnico
Suma con Acarreo	ADC
Suma	ADD
AND Lógica	AND
Bit de Prueba con el Acumulador	BIT
Compara el Acumulador con la Memoria	CMP
Compara el Registro de Índice con la Memoria	CPX
OR-Exclusiva de la Memoria con el Acumulador	EOR
Salto	JMP
Salto a Subrutina	JSR
Carga el Acumulador desde la Memoria	LDA
Carga el Registro de Índice desde la Memoria	LDX
OR-Inclusiva	ORA
Substracción con Acarreo	SBC
Guarda el Acumulador en la Memoria	STA
Guarda el Registro de Índice en la Memoria	STX
Substracción	SUB

Modo de Direccionamiento Directo

El modo de direccionamiento directo es similar al modo de direccionamiento extendido, excepto que se asume que el byte superior de la dirección del operando es $00. Así, sólo se necesita ser incluido en la instrucción el byte más bajo de la dirección del operando.

El direccionamiento directo permite direccionar efectivamente a los 256 bytes más bajos de la memoria. Esta área de memoria se llama 'página directa' e incluye la RAM interna y los registros de E/S. El direccionamiento directo es efectivo en memoria y tiempo. Las instrucciones de modo de direccionamiento directo normalmente tienen dos bytes, uno para el 'opcode' y otro para el byte de la parte más baja de la dirección del operando.

Listado de Programa Ejemplo:

```
0300    B6 50   LDA $50         ;Carga el acumulador desde la dirección directa
```

Secuencia de ejecución:
$0300 $B6 [1]
$0301 $50 [2] y [3]

Explicación:
[1] La CPU lee el 'opcode' $B6, carga el acumulador usando el modo de direccionamiento directo.
[2] Entonces la CPU lee $50 de la posición $0301. Este $50 es interpretado como la parte baja de la mitad de una dirección. En modo de direccionamiento directo, la parte alta de la mitad de la dirección es asumida para ser $00.

[3] La CPU añade internamente $00 a los $50 leídos por el segundo ciclo, para formar la dirección completa ($0050). La CPU entonces lee que valor que contiene en la posición $0050 en el acumulador.

A continuación se muestra una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento directo.

Instrucción	Código mnemotécnico
Suma con Acarreo	ADC
Suma	ADD
AND Lógica	AND
Desplazamiento aritmético a la izquierda	ASL
Desplazamiento aritmético a la derecha	ASR
Pone a 0 un Bit en la Memoria	BCLR
Prueba de Bit de la Memoria con el Acumulador	BIT
Bifurcación si el Bit n está a 0	BRCLR
Bifurcación si el Bit n está a 1	BRSET
Pone a 1 un Bit en la Memoria	BSET
Pone a 0	CLR
Compara el Acumulador con la Memoria	CMP
Complemento	COM
Compara el Registro de Índice con la Memoria	CPX
Decrementa	DEC
OR-Exclusiva de la Memoria con el Acumulador	EOR
Incrementa	INC
Salto	JMP
Salto a Subrutina	JSR
Carga el Acumulador desde la Memoria	LDA
Carga el Registro de Índice desde la Memoria	LDX
Desplazamiento Lógico a la Izquierda	LSL
Desplazamiento Lógico a la Derecha	LSR
Negación	NEG
OR-Inclusiva	ORA
Rotación a la Izquierda con Acarreo	ROL
Rotación a la Derecha con Acarreo	ROR
Substracción con Acarreo	SBC
Guarda el Acumulador en la Memoria	STA
Guarda el Registro de Índice en la Memoria	STX
Substracción	SUB
Prueba para Negativo o Cero	TST

Modos de direccionamiento Indexado

En el modo de direccionamiento indexado, la dirección efectiva es variable y depende en dos factores:

1. Los contenidos actuales del registro de índice (X)
2. El desplazamiento contenido en el byte(s) que sigue(n) al 'opcode'

Existen tres tipos de direccionamiento indexado en la MCU:

- Ningún desplazamiento
- Desplazamiento de 8-bits
- Desplazamiento de 16-bits

Un buen ensamblador debe usar el modo de direccionamiento indexado que exige el menor número de bytes para expresar el desplazamiento.

Modo Indexado sin Ningún Desplazamiento

En el modo de direccionamiento indexado sin ningún desplazamiento, la dirección efectiva de la instrucción está contenida en el registro de índice de 8-bits. Así que este modo de direccionamiento puede acceder a las primeras 256 posiciones de la memoria. Estas instrucciones son sólo de un byte.

Listado de Programa Ejemplo:

```
0300    F6      LDX ,x          ;Carga el acumulador desde la posición
                                ;apuntada al registro de índice (sin ningún desplazamiento)
```

Secuencia de la ejecución:
$0300 $F6 [1], [2], [3]

Explicación:
[1] La CPU lee el 'opcode' $F6, carga el acumulador usando el modo de direccionamiento indexado sin ningún desplazamiento.
[2] La CPU forma una dirección completa agregando $0000 a los contenidos del registro de índice.
[3] Entonces la CPU lee los contenidos de la posición direccionada en el acumulador.

A continuación se muestra una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado sin ningún desplazamiento o el modo de direccionamiento indexado con un desplazamiento de 8-bits.

Instrucción	Código mnemotécnico
Suma con Acarreo	ADC
Suma	ADD
AND Lógica	AND
Desplazamiento aritmético a la izquierda	ASL
Desplazamiento aritmético a la derecha	ASR
Prueba de Bit de la Memoria con el Acumulador	BIT
Pone a 0	CLR
Compara el Acumulador con la Memoria	CMP
Complemento	COM
Compara el Registro de Índice con la Memoria	CPX
Decrementa	DEC
OR-Exclusiva de la Memoria con el Acumulador	EOR
Incrementa	INC
Salto	JMP
Salto a Subrutina	JSR
Carga el Acumulador desde la Memoria	LDA
Carga el Registro de Índice desde la Memoria	LDX
Desplazamiento Lógico a la Izquierda	LSL
Desplazamiento Lógico a la Derecha	LSR
Negación	NEG
OR-Inclusiva	ORA
Rotación a la Izquierda con Acarreo	ROL
Rotación a la Derecha con Acarreo	ROR
Substracción con Acarreo	SBC
Guarda el Acumulador en la Memoria	STA
Guarda el Registro de Índice en la Memoria	STX
Substracción	SUB
Prueba para Negativo o Cero	TST

Indexado con 8-bits de Desplazamiento

En el modo de direccionamiento indexado con 8-bits de desplazamiento, la dirección efectiva se obtiene sumando los contenidos del byte que sigue al 'opcode' a los contenidos del registro de índice. Este modo de direccionamiento es útil para seleccionar el elemento enésimo en una tabla de 'n' elementos. Para usar este

modo, la tabla debe empezar en las 256 posiciones de memoria más bajas y puede extenderse a través de las primeras 511 posiciones de memoria (IFE es la última posición que la instrucción puede acceder). El modo de direccionamiento indexado con 8-bits de desplazamiento puede usarse para la ROM, RAM o E/S. Ésta es una instrucción de 2-bytes con el desplazamiento contenido en el byte que sigue al 'opcode'. El contenido del registro de índice (X) no cambia. El byte de desplazamiento proporcionado en la instrucción es un entero de 8-bits sin signo.

Listado de Programa Ejemplo:

```
0300    E6 05   LDA $5,x        ;Carga al acumulador desde la posición
                                ;apuntado por el registro de índice (X) + $05
```

Secuencia de la ejecución:

```
$0300   $E6     [1]
$0301   $05     [2],    [3],    [4]
```

Explicación:

[1] La CPU lee el 'opcode' $E6, carga el acumulador usando el modo de direccionamiento indexado con desplazamiento de 8-bits.

[2] Entonces la CPU lee $05 desde la posición $0301. Este $05 es interpretado como la parte baja de la mitad de una dirección base. La otra mitad de la parte alta se asume que la dirección base es $00.

[3] La CPU agregará el valor en el registro de índice de la dirección base $0005. El resultado de esta suma es la dirección que la CPU usará para cargar al acumulador.

[4] Entonces la CPU leerá el valor de esta dirección y cargará este valor en el acumulador.

Las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado con 8-bits de desplazamiento son iguales que la lista de instrucciones que pueden usar el modo de direccionamiento indexado sin ningún desplazamiento (véase la tabla anterior).

Indexado con 16-Bits de Desplazamiento

En el modo de direccionamiento indexado con 16-bits de desplazamiento, la dirección efectiva es la suma de los contenidos del registro índice de 8-bits y los dos bytes que siguen al 'opcode'. El contenido del registro de índice no se cambia. Estas instrucciones tienen tres bytes, uno para los 'opcode' y dos para un desplazamiento de 16-bits.

Listado de Programa Ejemplo:

```
0300    D6 07 00        LDA $0700,x     ;Carga el acumulador desde la posición
                                        ;apuntada en el registro de índice (X) + $0700
```

Secuencia de la ejecución:

```
$0300   $D6     [1]
$0301   $07     [2]
$0302   $00     [3],    [4],    [5]
```

Explicación:

[1] La CPU lee el 'opcode' $D6, carga el acumulador usando el modo de direccionamiento indexado con 16-bits de desplazamiento.

[2] Entonces la CPU lee $07 de la posición $0301. Este $07 es interpretado como la parte alta de la mitad de una dirección base.

[3] Entonces la CPU lee $00 de la posición $0302. Este $00 es interpretado como la parte baja de la mitad de una dirección base.

[4] La CPU agregará el valor en el registro de índice a la dirección base $0700. Los resultados de esta suma son la dirección que la CPU usará en la carga del acumulador.

[5] La CPU leerá el valor de esta dirección y cargará este valor en el acumulador.

A continuación se muestra una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado con 16-bits de desplazamiento.

Instrucción	Código mnemotécnico
Suma con Acarreo	ADC
Suma	ADD
AND Lógica	AND
Prueba de Bit de la Memoria con el Acumulador	BIT
Compara el Acumulador con la Memoria	CMP
Compara el Registro de Índice con la Memoria	CPX
OR-Exclusiva de la Memoria con el Acumulador	EOR
Salto	JMP
Salto a Subrutina	JSR
Carga el Acumulador desde la Memoria	LDA
Carga el Registro de Índice desde la Memoria	LDX
OR-Inclusiva	ORA
Substracción con Acarreo	SBC
Guarda el Acumulador en la Memoria	STA
Guarda el Registro de Índice en la Memoria	STX
Substracción	SUB

Modo de Direccionamiento Relativo

El modo de direccionamiento relativo sólo se usa para las instrucciones 'branch' (bifurcación). Las instrucciones 'branch', también utilizadas en instrucciones de manipulación de bit, generan dos bytes de código máquina: uno para el 'opcode' y otro para el desplazamiento relativo. Porque es deseable bifurcar en cualquier dirección, el byte del desplazamiento es un desplazamiento complemento a dos con signo con un rango de –127 a +128 bytes (con respecto a la dirección de la instrucción que sigue a la inmediata instrucción de bifurcación). Si la condición bifurcación es verdadera, los contenidos de los 8-bits siguientes del byte con signo, el 'opcode' (desplazamiento) se suma a los contenidos del contador de programa para formar la dirección de bifurcación efectiva; por otra parte, si la condición es falsa se procede a ejecutar la instrucción que sigue inmediatamente a la instrucción bifurcación.

Un programador especifica el destino de una bifurcación como una dirección absoluta (o etiqueta que se refiere a una dirección absoluta). El ensamblador de Motorola calcula el desplazamiento relativo de 8-bits con signo que se pone después del 'opcode' bifurcación en la memoria.

Listado de Programa Ejemplo:

```
0300    27 rr    BEQ    DEST    ;Bifurcación a DEST si Z = 1
                                ; (bifurcación si es igual o cero)
```

Sucesión de la ejecución:
$0300 $27 [1]
$0301 $rr [2], [3]

Explicación:
[1] La CPU lee el 'opcode' $27, bifurcación si Z = 1, (modo de direccionamiento relativo).
[2] La CPU lee el desplazamiento, $rr.
[3] La CPU prueba internamente el estado del bit Z y provoca una bifurcación si Z es 1.

A continuación se muestra una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento relativo.

Instrucción	Código mnemotécnico
Bifurca si el Acarreo es 0	BCC
Bifurca si el Acarreo es 1	BCS
Bifurca si es Igual	BEQ
Bifurca si Medio Acarreo es 0	BHCC
Bifurca si Medio Acarreo es 1	BHCS
Bifurca si es Mayor	BHI

Bifurca si es Mayor o Igual	BHS
Bifurca si la Línea de Interrupción es 1	BIH
Bifurca si la Línea de Interrupción es 0	BIL
Bifurca si es Menor	BLO
Bifurca si es Menor o Igual	BLS
Bifurca si el Bit de Máscara de Interrupción es 0	BMC
Bifurca si es Menos	BMI
Bifurca si el Bit de Máscara de Interrupción es 1	BMS
Bifurca si No es Igual	BNE
Bifurca si es Más	BPL
Bifurca Siempre	BRA
Bifurca si el Bit n es 0	BRCLR
Bifurca si el Bit n es 1	BRSET
Nunca Bifurca	BRN
Bifurca a Subrutina	BSR

Instrucciones Prueba de bit (Bit Test) y Bifurcación (Branch)

Estas instrucciones usan el modo de direccionamiento directo para especificar la posición probada y el direccionamiento relativo para especificar el destino de la bifurcación. Este libro trata estas instrucciones como instrucciones de modo de direccionamiento directo. Algún documento más viejo de Motorola llama el modo de direccionamiento de estas instrucciones BTB para la prueba de bit y bifurcación.

Organización de las instrucciones por Tipo

De la Tabla 11 hasta la Tabla 14 se muestra el juego de instrucciones del MC68HC05 desplegado por tipo de instrucción.

		Modos de Direccionamiento																	
		Inmediato			Directo			Extendido			Indexado (sin Offset)			Inexado (8 bits de Offset)			Indexado (16 bits de Offset)		
Función	Mnem.	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos
Carga A desde la memoria	LDA	A6	2	2	B6	2	3	C6	3	4	F6	1	3	E6	2	4	D6	3	5
Carga X desde la memoria	LDX	AE	2	2	BE	2	3	CE	3	4	FE	1	3	EE	2	4	DE	3	5
Guarda A en la memoria	STA	-	-	-	B7	2	4	CF	3	5	F7	1	4	E7	2	5	D7	3	6
Guarda X en la memoria	STX	-	-	-	BF	2	4	CF	2	5	FF	1	3	EF	2	5	DF	3	6
Suma la memoria a A	ADD	AB	2	2	BB	2	3	CB	2	5	FB	1	3	EB	2	4	DB	3	5
Suma la memoria y acarreo en A	ADC	A9	2	2	B9	2	3	C9	2	5	F9	1	3	E9	2	4	D9	3	5
Resta la memoria	SUB	A0	2	2	B0	2	3	C0			F0	1	3	E0	2	4	D0	3	5
Resta la memoria A con acarreo	SBC	A2	2	2	B2	2	3	C2			F2	1	3	E2	2	4	D2	3	5
AND de la memoria a A	AND	A4	2	2	B4	2	3	C4	2	5	F4	1	5	E4	2	4	66	3	5
OR de la memoria con A	ORA	AA	2	2	BA	2	3	CA	3	4	FA	1	3	EA	2	4	DA	3	5
Desplazamiento Lógico Derecho	EOR	A8	2	2	B8	2	3	C8	3	4	F8	1	3	E8	2	4	D8	3	5
Comparación Aritmética de A con la memoria	CMP	A1	2	2	E1	2	3	C1	3	4	F1	1	3	E1	2	4	D1	3	5
Comparación Aritmética de X con la memoria	CPX	A3	2	2	B3	2	3	C3	3	4	F3	1	3	E3	2	4	D3	3	5
Prueba Bit mem. con A. Comprar. Lógica	BIT	A5	2	2	B5	2	3	C5	3	4	F5	1	3	E5	2	4	D5	3	5
Salto Incondicional	JMP	-	-	-	BC	2	2	CC	3	3	FC	1	2	EC	2	3	DC	3	4
Salto a Subrutina	JSR	-	-	-	BD	2	5	CD	3	6	FD	1	5	ED	2	6	DD	3	7

Tabla 11. Instrucciones Registro/Memoria.

		Modos de Direccionamiento														
		Inerente (A)			Inerente (X)			Directo			Indexado (sin Offset)			Indexado (8 bits de Offset)		
Función	Mnem.	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos	Opcode	Bytes	Ciclos
Incrementa	INC	4C	1	3	5C	1	3	3C	2	5	7C	1	5	6C	2	6
Decrementa	DEC	4A	1	3	5A	1	3	3C	2	5	7A	1	5	6A	2	6
Borra	CLR	4F	1	3	5F	1	3	3F	2	5	7F	1	5	6F	2	6
Complemeta	COM	43	1	3	53	1	3	33	2	5	73	1	5	63	2	6
Negación complemento a dos	NEG	40	1	3	50	1	3	30	2	5	70	1	5	60	2	6
Rotación con desplazamiento a la Izquierda	ROL	49	1	3	59	1	3	39	2	5	79	1	5	69	2	6
Rotación con desplazamiento a la Derecha	ROR	46	1	3	56	1	3	36	2	5	76	1	5	66	2	6
Desplazamiento Lógico a la Izquierda	LSL	48	1	3	58	1	3	38	2	5	78	1	5	68	2	6
Desplazamiento Lógico a la Derecha	LSR	44	1	3	54	1	3	34	2	5	74	1	5	64	2	6
Desplazamiento Lógico a la Derecha	ASH	47	1	3	57	1	3	37	2	5	77	1	4	67	2	
Prueba para Negativo o Cero	TST	4D	1	3	5D	1	3	3D	2	4	7D	1	4	6D	2	5
Multiplica	MUL	42	1	11	-	-	-	-	-	-	-	-	-	-	-	-
Pone Bit a 0	BCLR	-	-	-	-	-	-	Ver Nota	2	5	-	-	-	-	-	-
Pone Bit a 1	BSET	-	-	-	-	-	-	Ver Nota	2	5	-	-	-	-	-	-

NOTA: Al contrario de otras instrucciones de leer - modificar - escribir, BCLR y BSET usan sólo el direccionamiento directo.

Tabla 12. Instrucciones de Lectura/Modificación/Escritura.

Modo de Direccionamiento Relativo Función	Código Mnemotécnico	Opcode	# Bytes	# Ciclos
Bifurca Siempre	BRA	20	2	3
Nunca Bifurca	BRN	21	2	3
Bifurca si es Mayor	BHI	22	2	3
Bifurca si es Menor o Igual	BLS	23	2	3
Bifurca si el Acarreo es 0	BCC	24	2	3
Bifurca si el Acarreo es 1	BCS	25	2	3
Bifurca si es Mayor o Igual	BHS	24	2	3
Bifurca si es Menor	BLO	25	2	3
Bifurca si No es Igual	BNE	26	2	3
Bifurca si es Igual	BEQ	27	2	3
Bifurca si Medio Acarreo es 0	BHCC	28	2	3
Bifurca si Medio Acarreo es 1	BHCS	29	2	3
Bifurca si es Más	BPL	2A	2	3
Bifurca si es Menos	BMI	2B	2	3
Bifurca si el Bit de Máscara de Interrupción es 0	BMC	2C	2	3
Bifurca si el Bit de Máscara de Interrupción es 1	BMS	2D	2	3
Bifurca si la Línea de Interrupción es 0	BIL	2E	2	3
Bifurca si la Línea de Interrupción es 1	BIH	2F	2	3
Bifurca a Subrutina	BSR	AD	2	6

Tabla 13. Instrucciones Bifurcación

Modo de Direccionamiento Relativo Función	Código Mnemotécnico	Opcode	# Bytes	# Ciclos
Transfiere de A a X	TAX	97	1	2
Transfiere de X a A	TXA	9F	1	2
Pone a 1 el Bit de Acarreo	SEC	99	1	2
Pone a 0 el Bit de Acarreo	CLC	98	1	2
Pone a 1 el Bit de Máscara de Interrupción	SEI	9B	1	2
Pone a 0 el Bit de Máscara de Interrupción	CLI	9A	1	2
Interrupción por Software	SWI	83	1	10
Retorno de Subrutina	RTS	81	1	6
Retorno de Interrupción	RTI	80	1	9
Reset al Puntero de Pila	RSP	9C	1	2
Ninguna operación	NOP	9D	1	2
Parada	STOP	8E	1	2
Espera	WAIT	8F	1	2

Tabla 14. Instrucciones de Control

Resumen del Juego de Instrucciones

Los microcontroladores usan un código de operación o 'opcode' para dar las instrucciones a la CPU. El juego de instrucciones para una CPU específica es el juego de todos los 'opcodes' que la CPU sabe ejecutar. Por ejemplo, la CPU MC68HC705J1A puede entender 62 instrucciones básicas, algunas de ellas tienen muchas variaciones que requieren 'opcode's separados. El juego de instrucciones de la familia MC68HC05 incluyen 210 únicas instrucciones 'opcodes'.

A continuación se muestra un listado alfabético de las instrucciones de la familia M68HC05 disponible por el usuario y la tabla de símbolos utilizados:

Símbolos de Código de condición

H	Medio Acarreo (Bit 4)	–	No Afectado
I	Máscara de Interrupción (Bit 3)	0	Puesto a 0
N	Negación (Bit 2 Signo)	1	Puesto a 1
Z	Cero (Bit 1)	×	Prueba y Pone a 1 si es verdadero,
C	Carry/Borrow (Bit 0)		si no lo pone a 0

Operadores Booleanas

( )	Contenidos de M, por ejemplo (M) significa el contenido de la posición de Memoria M	+	OR Inclusiva
		⊕	OR Exclusiva
—	NOT	-	Negación, complemento a dos
←	es cargado con …	•	AND Lógico
x	Multiplicación		

Registros de la CPU

A	Acumulador	PC	Contador de Programa
ACCA	Acumulador	PCH	Byte Alto del PC
CCR	Registro de Código de Condición	PCL	Byte Bajo del PC
X	Registro de Índice	SP	Indicador de la Pila
M	Cualquier posición de memoria	REL	Dirección Relativa, Un Byte

Modos de Direccionamiento	**Abreviación**	**Operandos**
Inherente	INH	ninguno
Inmediato	IMM	ii
Directo (Para Instrucciones	DIR	dd
Prueba de Bit)		dd rr
Extendido	EXT	hh ll
Indexado sin ningún Desplazamiento	IX	ninguno
Indexado con 1-Byte de Desplazamiento	IX1	ff
Indexado con 2-Bytes de Desplazamiento	IX2	ee ff
Relativo	REL	rr

Resumen

Registros de la CPU

Los cinco registros de la CPU en la familia MC68HC05 no son posiciones en el mapa de memoria. El modelo de programación para la CPU muestra los cinco registra de la CPU.

- **Acumulador** (A) es un registro de 8-bits de propósito general.
- **Registro de índice** (X) es un registro de 8-bits del puntero.
- **Indicador de pila** (SP) es un registro indicador que es decrementado automáticamente cuando los datos se envían hacia la pila y se incrementa cuando los datos sacan fuera de la pila.
- **Contador de programa** (PC) tiene tantos bits como líneas de dirección. El contador de programa siempre apunta a la siguiente instrucción o bit de datos que la usará la CPU.
- **Registro de código de condición** (CCR) contiene las banderas de cuatro resultados aritméticos H, N, Z y C; y el control del bit I de máscara de interrupción (desactivada).

Modos de Direccionamiento

La CPU MC68HC05 tiene seis modos de direccionamiento, que determinan cómo la CPU conseguirá los operando(s) necesarios para completar cada instrucción. La CPU MC68HC05 tiene sólo 62 instrucciones mnemónicas. Hay 210 instrucciones 'opcodes' porque cada diferente modo de direccionamiento de una instrucción tiene un único 'opcode'.

- Modo de direccionamiento **inmediato**, el operando para la instrucción es el byte inmediatamente siguiente al 'opcode'.
- Modo de direccionamiento **inherente**, la CPU no necesita ningún operando de la memoria. Los operandos, si hay alguno, son los valores de los registros o los datos de la pila.
- Modo de direccionamiento **extendido**, los 16-bits de direcciones del operando, se localizan en los siguientes dos bytes de memoria después de la instrucción 'opcode'.
- Modo de direccionamiento **directo**, los 8 bits de la parte baja de la dirección del operando se localiza en el próximo byte de memoria después del 'opcode' y el byte de la parte alta de la dirección se asume como $00. Este modo es más efectivo que el modo de direccionamiento extendido, porque el byte de dirección de la parte alta no es explícitamente incluido en el programa.
- Modo de direccionamiento **indexado**, el valor actual del registro de índice se suma a un desplazamiento de 0, 1 o 2-bytes en las siguientes posiciones de memoria después del 'opcode' para formar un indicador de la dirección del operando en memoria.
- Modo de direccionamiento **relativo**, se usa para instrucciones de bifurcación condicional. El byte después del 'opcode' es un valor de desplazamiento con signo entre –128 y +127. Si la condición de bifurcación es verdad, el desplazamiento se suma al valor del contador de programa para conseguir la dirección donde la CPU sacará la siguiente instrucción del programa.

Ejecución de Instrucciones

Cada 'opcode' le dice a la CPU la operación a ser realizada y el modo de direccionamiento es usado para direccionar cualquier operandos necesitado para completar la instrucción. Las explicaciones ciclo a ciclo de las instrucciones del ejemplo bajo cada modo de direccionamiento, proporcionan una vista de los diminutos pasos simples que constituyen una instrucción.

Programación

Índice

Introducción

Este capítulo describe la manera de planificar y escribir programas. Se aprende a preparar organigramas, escribir programas en lenguaje ensamblador y usar el editor de texto para escribir programas. Después, se usa una herramienta de programación llamada 'Ensamblador', que traduce el programa de tal forma que el microcontrolador lo puede usar. Las herramientas de programación, son los programas de PC que ayudan al desarrollo de programas para microcontroladores. Se describen los ensambladores, simuladores y otras herramientas de desarrollo útiles.

Escribiendo un simple programa

A continuación se escribe un corto programa, en forma de código mnemotécnico y se traduce en código máquina. Éstos son los pasos:

- El primer paso es planificar el programa y documentarlo, con un organigrama.
- Seguidamente, se escriben los mnemónicos de cada instrucción para cada bloque del organigrama.
- Finalmente, se usa un 'Ensamblador' para traducir el programa ejemplo en códigos que el microcontrolador necesita para ejecutar el programa.

El programa leerá el estado de un pulsador conectado a un pin de entrada. Cuando el pulsador está cerrado, el programa hará que un LED conectado a un pin de salida se encienda aproximadamente durante un segundo y después se apague. El LED no se encenderá de nuevo hasta que el pulsador se haya dejado de pulsar y se cierre de nuevo. La duración en que el pulsador permanecerá cerrado, no afectará a la duración en que el LED estará encendido.

Aunque este programa es muy simple, muestra un elemento muy común, el LED, que está presente en cualquier programa de aplicación:

- Primero, muestra cómo un programa puede detectar las señales de entrada, tales como las de un interruptor que conmuta.
- Segundo, éste es un ejemplo de un programa que controla una señal de salida.
- Tercero, muestra una manera de que un programa se puede usar para medir tiempo real, controlando el tiempo de encendido de un LED durante un segundo aproximadamente.

Ya que el algoritmo es suficientemente complicado, no se puede lograr de una manera trivial con componentes discretos. Como mínimos, serían necesario unos circuitos integrados con componentes externos de temporización. Este ejemplo demuestra que un microcontrolador y un programa definido por un usuario (software), puede reemplazar varios circuitos complejos.

Diagrama de flujo

La Figura 30 muestra un organigrama o **diagrama de flujo** de un programa ejemplo. Los diagramas de flujo son a menudo usados como herramientas de planificación para escribir un programa (software), porque muestran la función y flujo del programa en desarrollo, siendo de suma importancia las anotaciones, los comentarios y la documentación incluida. De la misma manera de que el diseño de un circuito impreso no se considera completo sin un esquema, una lista de componentes y un dibujo del montaje, no se debe considerar un programa completo si no se tiene un listado comentado y una explicación comprensiva del programa, como la de un organigrama.

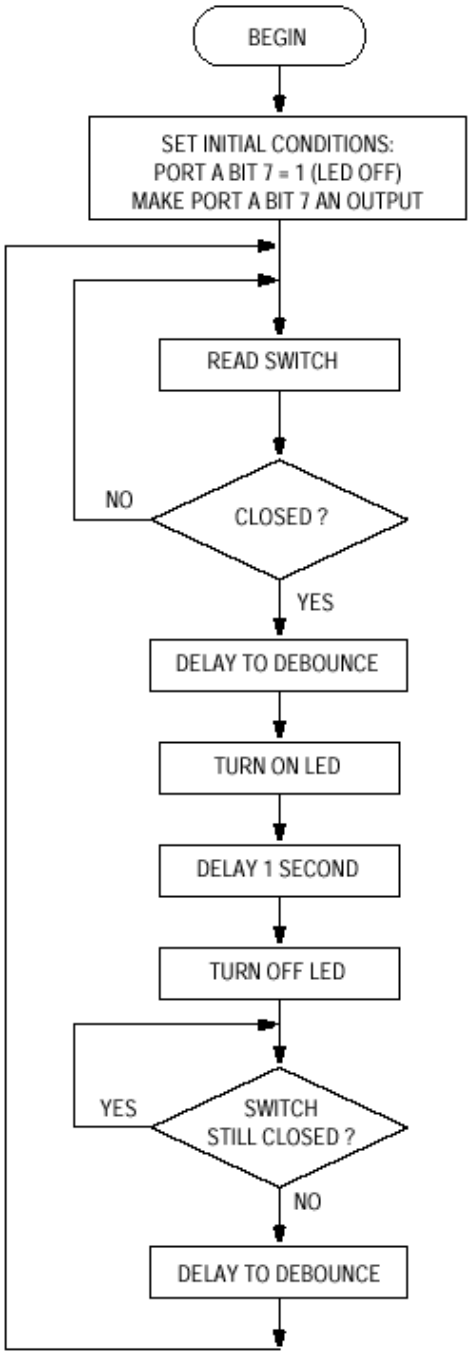


Figura 30. Diagrama de flujo del Ejemplo.

Código Fuente Mnemónico

Una vez se completa el diagrama de flujo, el programador desarrolla una serie de instrucciones en **lenguaje ensamblador** para lograr las funciones requeridas en cada bloque. El programador de software se limita a seleccionar las instrucciones de la CPU que va a usar (en este caso el MC68HC05) y las escribe en forma de mnemónicos que es fácil de entender. La Figura 31 muestra el **código fuente** mnemónico al lado del diagrama de flujo del programa ejemplo, para que se puedan ver las instrucciones que se usan para lograr cada uno de los bloques del diagrama de flujo. Los significados de los mnemónicos usados en el lado derecho de la figura 31 se pueden encontrar en el capítulo **Detalles del Juego de Instrucciones** o en el **Resumen del Juego de Instrucciones**.

Durante el enunciado del programa, se ha expuesto que se necesita un retardo en tres lugares distintos. Para ello se ha de desarrollar una **subrutina** que genere un retardo de 50 ms. Esta subrutina se usa directamente en dos lugares (para los rebotes de un interruptor) y otra para hacer el retardo de 1 segundo. En la figura 31, se omiten los comentarios que normalmente serían incluidos dentro del programa fuente, como documentación. Se mostrarán los comentarios en el programa completo (**Listado 3. Listado Ensamblador**).

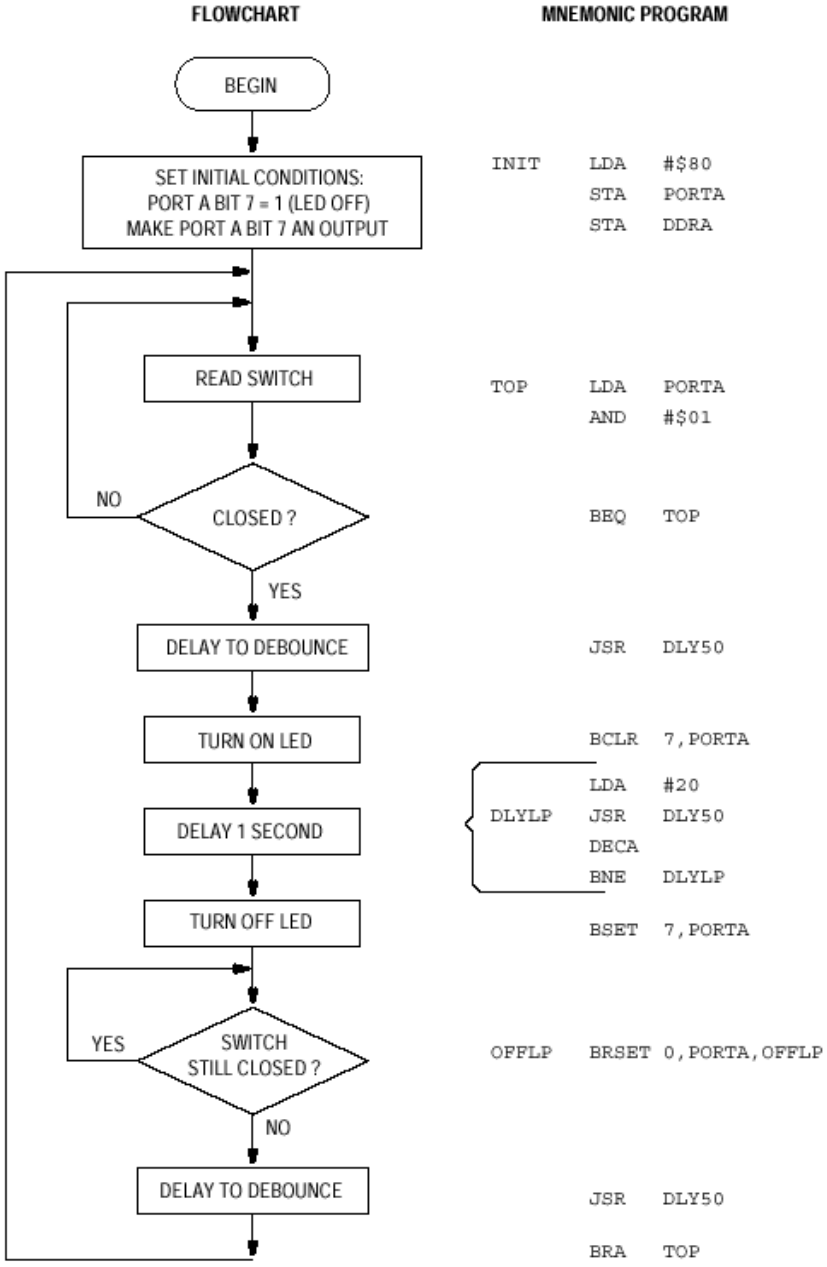


Figura 31. Organigrama y Mnemónicos.

Programa de Retardo

La Figura 32 muestra un organigrama extendido de la subrutina de retardo de 50 ms. Una subrutina es un programa relativamente corto que normalmente realiza algunas funciones requeridas. Cuando la función necesita ser realizada muchas veces en el curso de un programa, con la subrutina sólo se tiene que escribir una sola vez. Cada vez que se necesita esta función, el programador llama a la subrutina con una instrucción de 'bifurcación a subrutina' (BSR) o de 'salto a subrutina' (JSR).

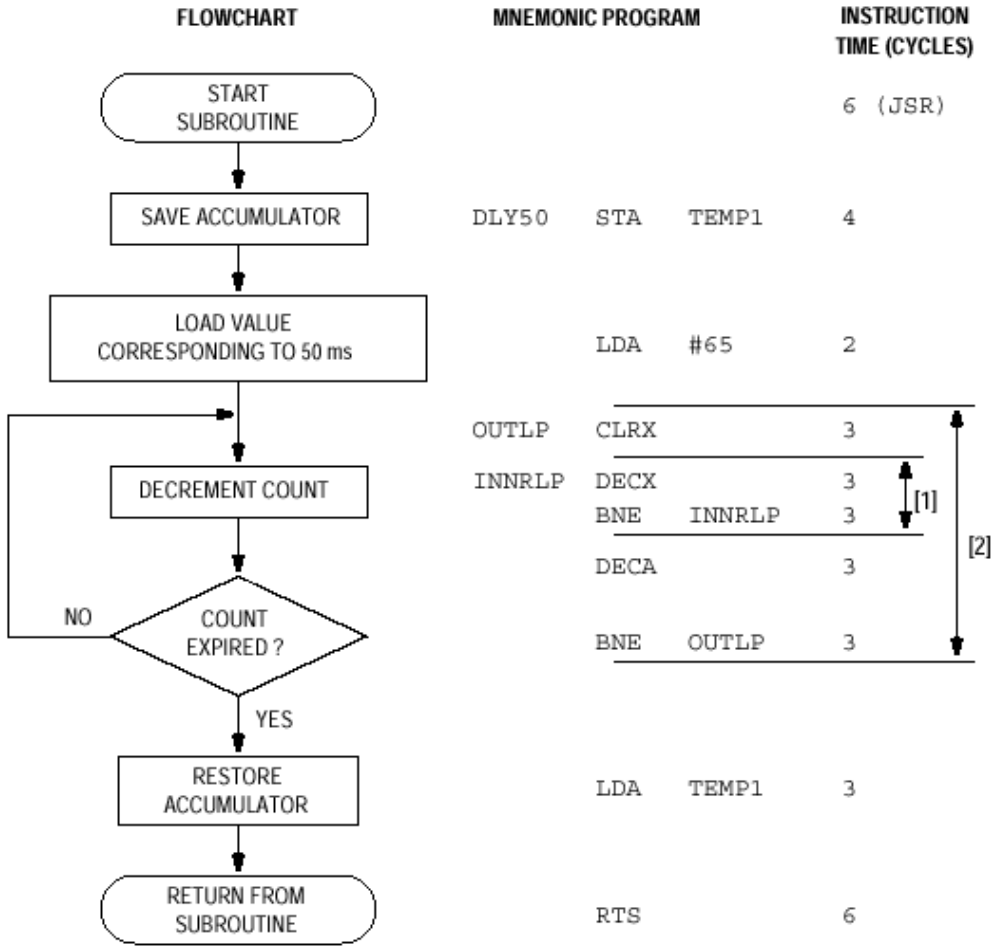


Figura 32. Organigrama de la Rutina de Retardo y mnemónicos

Antes de empezar a ejecutar las instrucciones en la subrutina, la dirección de la instrucción que sigue a JSR (o BSR) se guarda automáticamente en la pila, en las posiciones de memoria RAM temporales. Cuando la CPU acaba la ejecución de las instrucciones dentro de la subrutina, se realiza una instrucción de 'retorno de subrutina' (RTS), como última instrucción de la subrutina. La instrucción RTS causa que la CPU recupere la dirección de retorno previamente guardada; así, la CPU continúa el programa con la instrucción que sigue JSR (o BSR) que originalmente llamó a la subrutina.

La rutina de retardo de la Figura 32 involucra un lazo interno (INNRLP) dentro de otro lazo (OUTLP). El lazo interno consiste en dos instrucciones ejecutadas 256 veces antes de que X alcance $00 y acabe la condición de bifurcación BNE. Esto suma seis ciclos, a 500 ns por 256 ciclos, es igual a 0.768 ms para el lazo interno. El lazo externo se ejecuta 65 veces. El tiempo total de la ejecución para el lazo externo es 65 (1536+9) o 65 (1545) = 100,425 ciclos o 50.212 ms. Las diversas instrucciones de esta rutina son un total de 21 ciclos del lazo externo; así que, el tiempo total exigido para ejecutar la rutina DLY50 es 50.223 ms, incluyendo el tiempo requerido para la instrucción JSR que llama a la subrutina DLY50.

El sistema temporizador interno del MC68HC705J1A también se puede usar como una medida de tiempo. Es preferible este método basado en el temporizador, para que la CPU pueda realizar otras tareas durante el retardo y el tiempo de retardo no depende del número exacto de instrucciones ejecutadas, todo el contrario que en la subrutina DLY50.

Listado Ensamblador

Después de escribir un programa completo o una subrutina, se debe convertir de código mnemotécnico a código máquina (binario), para que después la CPU lo pueda ejecutar. Se utiliza un PC para realizar esta conversión a leguaje máquina, con un programa llamado ensamblador. El ensamblador lee el código mnemotécnico del programa (también llamado código fuente del programa) y produce código máquina del programa, en una forma que después se podrá programar en la memoria de la MCU.

El ensamblador también produce un listado compuesto de ambos, donde muestra el programa fuente original (mnemónicos) y el código objeto traducido. Este listado se usa durante la fase de depuración del proyecto y como parte de la documentación del programa (software). El **Listado 3. Listado Ensamblador** muestra el listado que resulta de ensamblar el programa ejemplo. Los comentarios se agregan antes de ensamblar el programa.

Listado 3. Listado Ensamblador

```
*******************************************************
* Programa ejemplo 68HC05
* Lee el estado de un interruptor en el bit 0 del puerto A; 1 = cerrado
* Cuando se cierra, el LED se enciende 1 segundo; El LED se enciende
* cuando el bit 7 del Puerto A es 0. Espera a que se abra el interruptor,
* y entonces repetir. Rebotes del interruptor de 50mS
* NOTA: Tiempos basados en el tiempo de ejecución de las instrucciones
* Si usa un simulador o cristal menor de 4MHz, esta rutina correrá más
* lenta que la deseada
*******************************************************
$BASE 10T          ; Díce al ensamblador que usa el modo decimal
                   ; menos los valores $ o %
0000          PORTA EQU $00      ;Dirección Directa del puerto A
0004          DDRA  EQU $04      ;Dirección control de dato, puerto A
00E0          TEMP1 EQU $C0      ;One byte temp storage location
0300                ORG $0300    ;El Programa empezará en $0300
0300 A6 80    INIT  LDA #$80     ;Empieza la inicialización
0302 B7 00          STA PORTA    ;El LED se apagará
0304 B7 04          STA DDRA     ;Pone el bit 7 del puerto como salida
* El resto del puerto A se configura como entrada
0306 B6 00    TOP   LDA PORTA    ;Lee el interruptor en LSB del Puerto A
0308 A4 01          AND #$01     ;Prueba el bit-0
030A 27 FA          BEQ TOP      ;Lazo hasta bit-0 = 1
030C CD 0323        JSR DLY50    ;Retardo de 50 ms para los rebotes
030F 1F 00          BCLR 7,PORTA ;Enciende el LED (bit-7 a 0)
0311 A6 14          LDA #20      ;El Decimal 20 se ensambla como $14
0313 CD 0323        DLYLP JSR DLY50 ;Retardo de 50 ms
0316 4A             DECA         ;Contador de Lazo hasta 20 lazos
0317 26 FA          BNE DLYLP    ;20 veces (20-19,19-18,...1-0)
0319 1E 00          BSET 7,PORTA ;Apaga el LED
031B 00 00FD OFFLP BRSET 0,PORTA,OFFLP ;Lazo hasta interruptor abierto
031E CD 0323        JSR DLY50    ;Retorno realizado
0321 20 E3          BRA TOP      ;Mira para el siguiente cierre del
interruptor
* DLY50 — Subrutina de retardo ~50ms
* Guarda el valor original del acumulador
* pero X siempre estará a cero en el retorno
0323 B7 C0 DLY50  STA TEMP1    ;Guarda el acumulador en la RAM
0325 A6 41        LDA #65      ;lazo externo de 65 veces
0327 5F   OUTLP   CLRX         ;X se usa como lazo interno contador
0328 5A   INNRLP  DECX         ;0-FF, FF-FE,...1-0 256 lazos
0329 26 FD        BNE INNRLP   ;6 ciclos * 256 * 500ns/ciclo = 0.768ms
032B 4A           DECA         ;65-64, 64-63,...1-0
032C 26 F9        BNE OUTLP    ;1545 ciclos * 65 * 500ns/ciclo = 50.212ms
032E B6 C0        LDA TEMP1    ;Recupera el valor guardado en el acumulador
0330 81           RTS          ;Retorno
```

Para la siguiente descripción hay que referirse a la Figura 33. Esta figura muestra algunas las líneas del listado con números de referencia indicando las varias partes de la línea. La primera línea es un ejemplo de una línea de directiva del ensamblador. Esta línea realmente no es parte del programa; más bien, proporciona información al ensamblador para que el programa real pueda convertirse propiamente en código máquina (binario).

```
0000                    PORTA EQU   $00                 ;Dirección directa del puerto A
0300                                ORG     $0300       ;El Programa empezará en $0300
0306    B6 00           TOP         LDA     PORTA       ;Lee pulsador en LSB del Puerto A
------- ------------------- --------- ---------- -------------------- ----------------------------------------------
--
[1]         [2]             [3]         [4]         [5]                 [6] →
```

Figura 33. Explicación del Listado Ensamblador

EQU, es la forma abreviada de 'equale' (igual), se usa para dar un nombre a una posición de memoria específica o a número binario, que puede usarse en otras instrucciones del programa. En este caso, la directiva EQU se usa para asignar el nombre PORTA al valor $00 que es la dirección del registro del puerto A en el MC68HC705J1A. Es más fácil para un programador recordar el nombre mnemónico PORTA en lugar del anónimo valor numérico $00. Cuando el ensamblador encuentra uno de éstos nombres, el nombre se reemplaza automáticamente por su correspondiente valor binario, de la misma manera por la que se reemplazan las instrucciones mnemónicas por instrucciones de códigos binarios.

La segunda línea mostrada en la Figura 33 es otra directiva del ensamblador. El mnemónico ORG, forma abreviada de 'originate' (origen), le dice al ensamblador donde empezará el programa (la dirección origen de la primera instrucción que sigue a la línea directiva ORG). ORG más que como una directiva, se puede usar en un programa para decirle al ensamblador donde poner partes diferentes del programa, en lugares específicos de la memoria. Referirse al mapa de memoria de la MCU para seleccionar una posición de memoria apropiada, de donde se debe empezar un programa.

En el listado ensamblador, los dos primeros campos [1] y [2], son generados por el ensamblador y los cuatro siguientes campos [3], [4], [5] y [6], son originales del programa fuente escrito por el programador. En el campo [3] es una etiqueta (TOP) qué puede enviarse a otras instrucciones. En el programa ejemplo, la última instrucción es BRA TOP, que simplemente significa que la CPU continuará la ejecución con la instrucción que se etiqueta TOP.

Cuando el programador está escribiendo un programa, las direcciones donde se localizarán las instrucciones típicamente no se conocen. Pero todavía es peor, en instrucciones de bifurcación, en lugar de usar la dirección de un destino, la CPU usa un desplazamiento (diferencia entre el valor actual y la dirección de destino). Afortunadamente, el programador no tiene que preocuparse por estos problemas porque el ensamblador cuida de estos detalles a través de un sistema de etiquetas. Este sistema de etiquetas es una buena manera para que el programador pueda identificar puntos específicos en el programa (sin conocer sus direcciones exactas); el ensamblador puede convertir después estas etiquetas mnemónicas en direcciones de memoria específicas e incluso calcular los desplazamientos para las instrucciones de bifurcación para que la CPU pueda usarlos.

El campo [4], es el campo de la instrucción. El código mnemotécnico LDA es la forma abreviada 'load accumulator' (cargar el acumulador). Ya que hay seis variaciones (opcodes diferentes) de la instrucción LDA, se requiere la información adicional, antes de que el ensamblador pueda escoger el 'opcode' binario correcto para que la CPU lo use durante la ejecución del programa.

El campo [5], es el campo del operando y proporciona información sobre la posición específica de la memoria o el valor que va a ser procesado por la instrucción. El ensamblador usa el código mnemotécnico de la instrucción y el operando especificado en el programa fuente, para determinar el 'opcode' específico para la instrucción.

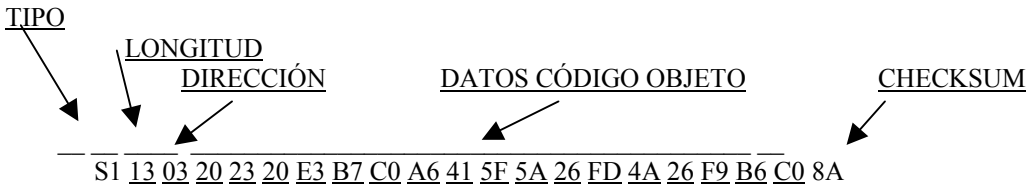
Hay diferentes modos de especificar como va ha ser procesado el valor, son los llamados modos de direccionamiento. (Una descripción más completa de los modos de direccionamiento fueron presentados en **Modos de Direccionamiento**). La sintaxis del campo del operando es ligeramente diferente para cada modo de direccionamiento, para que el ensamblador pueda determinar el modo de direccionamiento correcto deseado de la sintaxis del operando. En este caso, el operando [5] es PORTA, que el ensamblador automáticamente lo ha convertido en $00 (llamada a la directiva EQU). El ensamblador interpreta $00 como un modo de direccionamiento directo entre $0000 y $00FF, seleccionando así el 'opcode' $B6, que es el modo de direccionamiento directo, variación de la instrucción LDA. Si PORTA se hubiera precedido por un símbolo #, la sintaxis habría sido interpretada por el ensamblador como un valor del modo de direccionamiento inmediato y se habría escogido el 'opcode' $A6 en lugar de $B6.

El campo [6], es el campo de comentarios y no es usado por el ensamblador para traducirlo en código máquina. Más bien, el campo de comentarios es usado por el programador para documentar el programa. Aunque la CPU no use esta información durante la ejecución del programa, un programador, sabe que es una de las partes más importantes de un buen programa. Los comentarios [6] para esta línea de programa dice: *Lee el pulsador en LSB del puerto A*. Este comentario dice a alguien que está leyendo el listado, algo sobre la instrucción o por qué está allí, que es esencial para entender cómo trabaja el programa. El punto y coma indica que el resto de la línea debe tratarse como un comentario (no todos los ensambladores requieren este punto y coma). Puede hacerse una línea entera para una línea de comentario usando un asterisco (*) como primer carácter de la línea. Además de un listado con buenos comentarios, también es importante documentar los programas con un organigrama o con otra información detallada que explique el flujo global y el funcionamiento del programa.

Archivo de Código Objeto

En la **Arquitectura del Microprocesador**, se aprendió que el microprocesador espera el programa como una serie de valores de 8 bits en la memoria. Hasta ahora, este programa ejemplo todavía parece como si fuera escrito para los humanos. La versión que necesita cargar la MCU en su memoria se llama **archivo de código objeto**. Los microcontroladores de Motorola tienen una forma común del archivo de código objeto, es el **S-record**. Al ensamblador se le puede dar una opción para producir un archivo de listado y/o un archivo de código objeto. Un archivo S-record es un archivo de texto ASCII que puede ser visto por un editor de texto. Aunque, no se deben revisar estos archivos porque la estructura y el contenido de los archivos es crítico para su funcionamiento apropiado.

Cada línea de un archivo S-record es un **registro**. Cada registro empieza con una letra mayúscula S seguida por un número de código del 0 al 9. Únicamente algunos números de código son importantes al programador y son S0, S1 y S9, porque otros códigos S-número sólo se aplican a los sistemas más grandes. S0 es el registro de cabecera optativo, que puede contener el nombre del archivo para beneficio de los humanos, que necesitan para mantener estos archivos. Los registros S1 son los registros principales de los datos. Se usa un registro S9 para marcar el final del archivo S-record. Para el trabajo con microcontroladores de 8-bits, la información del registro S9, no es importante, pero es necesario un registro S9 al final de los archivos S-record. La Figura 34 muestra la sintaxis de un registro S1.



CHECKSUM = Complemento a uno de la suma de todos estos bytes

Figura 34. Sintaxis de un registro S1

Todos los números de un archivo S-record están en hexadecimal. Todo archivo S-record tiene diferentes campos. El campo **tipo**, es S0, S1 o S9. El campo **longitud**, es el número de pares de dígitos hexadecimales que hay en el registro, excluyendo los campos tipo y longitud. El campo **dirección**, es la dirección de 16 bits donde el primer byte de datos se guardará en la memoria. Cada par de dígitos hexadecimales en el campo **código objeto de datos**, representan un valor de datos de 8-bits para ser guardados en sucesivas posiciones en la memoria. El campo **checksum**, es un valor de 8-bits que representa los complementos a uno de la suma de todos los bytes del S-record, excepto los campos tipo y checksum. Este checksum se usa para verificar durante la carga del archivo S-record, que los datos están completos y correctos para cada registro.

La Figura 35 muestra un archivo S-record, que es el resultado de ensamblar el programa ejemplo **Listado 3. Listado Ensamblador**. Los dos bytes de código máquina de datos que están resaltados en negrita, son los mismos dos bytes en los que se resaltaron en la Figura 16 y el texto de la Figura 16. Estos bytes fueron localizados mirando en el listado y fijándose que la dirección donde esta instrucción empezada era $0323. En el archivo S-record, se encuentra el registro S1 con la dirección $0320. Mirando hacia la derecha, se encuentran los datos $23 para la dirección $0320 y $20 para la dirección $0321, $E3 para la dirección $0322 y finalmente los bytes que se querían encontrar para la dirección $0323 y $0324.

S1130300A680B700B704B600A40127FACD03231FC3
S113031000A614CD03234A26FA1E000000FDCD03D7
S11303202320E3**B7C0**A6415F5A26FD4A26F9B6C08A

```
S10403308147
S9030000FC
```

Figura 35. Archivo S-record del Programa Ejemplo

Directivas del Ensamblador

En esta sección se describen seis de las directivas del ensamblador más importantes. Las directivas que soportan los ensambladores de varios fabricantes, difieren en el número y en el tipo. Siempre hay que referirse a la documentación de cada ensamblador que se está usando.

Origen (ORG)

Esta directiva se usa para poner el contador de posición para el ensamblador. El contador de posición guarda el rastro de la dirección donde el próximo byte de código máquina se guardará en memoria. En el **programa ejemplo**, la directiva ORG pone el inicio del programa en $0300. Ensamblará a partir de esa dirección especificada. Cuando el ensamblador traduce declaraciones del programa a instrucciones y datos de código máquina, el contador de posición avanza para apuntar a la siguiente posición de memoria disponible.

Cada programa tiene por lo menos una directiva ORG para establecer el punto de inicio en la memoria, para el programa. La mayoría de los programas completos, también tendrá una segunda directiva ORG cerca del final del programa para poner la contador de posición en la dirección donde son localizados los vectores de interrupción y reset ($07F8–$07FF en el MC68HC705J1A). El vector de reset siempre se debe especificar y también es buena práctica especificar los vectores de interrupción, aun cuando no se esperan usar las interrupciones.

Igual (EQU)

Esta directiva se usa para asociar un valor binario con una etiqueta. El valor puede ser un valor de 8-bits o un valor de dirección de 16-bits. Este directiva no genera ningún código objeto.

Durante el proceso de ensamblar, el ensamblador debe guardar una tabla de referencia cruzada, donde se guardan los valores binarios equivalentes de cada etiqueta. Cuando una etiqueta aparece en el programa fuente, el ensamblador mira esta tabla de referencia cruzada para encontrar el valor binario equivalente. Cada directiva EQU genera una entrada en esta tabla de referencia cruzada.

Un ensamblador lee el programa fuente dos veces. En el primer paso, el ensamblador apenas cuenta los bytes de código objeto e internamente construye la tabla de referencia cruzada. En el segundo paso, el ensamblador genera el archivo de listado y/o el archivo objeto S-record. Este segundo paso permite al programador hacer referencia a las etiquetas, en las que se definen más tarde en el programa.

Las directivas EQU deben aparecer cerca del inicio de un programa, antes de que sus etiquetas sean usadas a través de otras declaraciones del programa. Si el ensamblador encuentra una etiqueta antes de que se definida, pero no tendrá ninguna elección pero para asumir, en el peor de los casos, de un valor de dirección de 16-bits. Esto provocará el uso del modo de direccionamiento extendido en lugares donde se podría usar el método más eficaz, el modo de direccionamiento directo. En otros casos, se puede usar el modo de direccionamiento indexado con un desplazamiento de 16-bits, donde se podría usar una instrucción indexada más eficaz con un desplazamiento de 8-bits o sin ningún desplazamiento.

En el programa ejemplo, hay dos directivas EQU para definir las etiquetas PORTA y DDRA a sus direcciones de página directa. Otro uso de las directivas EQU es identificar una posición de bit con una etiqueta:

```
LED     EQU     %10000000           ;El LED se conecta al bit-7
  “       “         “                       “
  “       “         “                       “
INIT    LDA     #LED                ;Hay un 1 en el bit de posición LED
        STA     PORTA               ;el LED se apagará
        STA     DDRA                ;el pin LED es una salida
```

El símbolo % indica que el valor que sigue se expresa en binario. Si el LED se mueve a un pin diferente durante el desarrollo, sólo se tendrá que cambiar la declaración EQU y volver a ensamblar el programa.

Byte de Forma Constante (FCB)

Los argumentos para esta directiva son etiquetas o números, separadas o separados por comas, que pueden convertirse en simples bytes de datos. Cada byte especificado en una directiva FCB genera un byte de

código máquina en el archivo de código objeto. Se usan directivas FCB para definir las constantes en un programa.

Byte de Forma Doble (FDB)

Los argumentos para esta directiva son etiquetas o números, separadas o separados por comas, que pueden convertirse en valores de datos de 16-bits. Cada argumento especificado en una directiva FDB genera dos bytes de código máquina en el archivo de código objeto.

Éstas líneas de listado ensamblador muestra las directivas ORG y FDB

```
“    “  “    “          “     “              “
“    “  “    “          “     “              “
0300                    ORG   $0300          ;Inicio de la EPROM en el 705J1A
0300 B6 00 START        LDA   PORTA          ;Lee el pulsador en LSB del puerto A
“    “  “    “          “     “              “
“    “  “    “          “     “              “
041F 80    UNUSED       RTI                  ;Vuelve de la interrupción inesperada
“    “  “    “          “     “              “
“    “  “    “          “     “              “
07F8                    ORG   $07F8          ;Inicio del área de vectores
07F8 04 1F TIMVEC       FDB   UNUSED         ;Un vector sin usar
07FA 04 1F IRQVEC       FDB   $041F          ;El argumento puede ser un valor hex
07FC 04 1F SWIVEC       FDB   UNUSED         ;Un vector sin usar
07FE 03 00 RESETV       FDB   START          ;Ir a START si hay un ‘reset’
```

Byte de Reserva de Memoria (RMB)

Esta directiva se usa para poner espacio adicional en la RAM, para las variables del programa. La directiva RMB normalmente no genera código objeto, pero genera una entrada en la tabla de referencia cruzada interna del ensamblador. En el programa ejemplo (**Listado 3. Listado Ensamblador**), la RAM variable fue asignada como TEMP1 con una directiva EQU.

Otra manera de asignar esta variable, es así:

```
“    “  “    “          “     “              “
00C0                    ORG   $00C0          ;Inicio de la RAM en el 705J1A
00C0       TEMP1        RMB   1             ;1 byte en la posición de almacenamiento temporal
“    “  “    “          “     “              “
```

Ésta es la manera preferida para asignar el almacenamiento de la RAM, porque es común agregar y anular variables en el curso del desarrollo de un programa. Si se usan directivas EQU, después se podría tener que cambiar varias declaraciones quitando una sola variable. Con las directivas RMB, el ensamblador asigna direcciones cuando las necesita.

Pone el Número Base por defecto a Decimal

Algunos ensambladores, como el ensamblador IASM de P&E Microcomputer Systems, asumen que cualquier valor que no se marca específicamente, se debe interpretar como un valor hexadecimal. La idea es simplificar la entrada de información numérica, eliminando la necesidad de poner el símbolo $ antes de cada valor. Si se quiere que el ensamblador asuma que estos valores sin marca, sean números decimales, se debe usar la directiva $BASE.

```
“    “  “    “          “     “              “
. . . .    $BASE        10T                  ;Pone por defecto la base # a decimal
000A       TEN          EQU   #10            ;10 decimal no es $10 = 16
```

Esta directiva es ligeramente diferente de las otras descritas en este capítulo. La directiva $BASE empieza en la columna de más a la izquierda del programa fuente. Esta directiva se incluye cerca de la de inicio de cada programa ejemplo. Si se está usando un ensamblador que no requiere este directiva, se puede anular o se puede agregar un asterisco (*) en la línea de inicio para hacer un comentario de la línea. Cuando se comenta una línea fuera del programa, se cambia la línea entera con un comentario. Los comentarios no afectan al ensamblado de un programa.

Familiarización con el Juego de Instrucciones

Como en la mayoría de los campos de la ingeniería, más de una sucesión de instrucciones, puede realizar cualquier tarea. Una buena manera de aprender un nuevo juego de instrucciones, es ver de cuántas maneras diferentes se puede resolver algún pequeño problema de programación. A esto se le llama destreza del juego de instrucciones.

La Figura 36 muestra cuatro maneras diferentes para verificar el cierre de un interruptor conectado al bit 0 del puerto A. Se usaron dos de estas maneras en el programa ejemplo del **Listado 3. Listado Ensamblador**. Aunque todas las secuencias logran la misma tarea básica, hay sutiles diferencias. Normalmente estas diferencias no son significantes, pero a veces pueden ahorrar tiempo de ejecución o espacio de memoria de programa. En un microcontrolador pequeño, el espacio de memoria puede ser una consideración importante.

```
0000                    PORTA   EQU     $00             ;Dirección directa del puerto A
0300                            ORG     $0300           ;El programa empezará en la dirección $0300
0300  B6 00     [3]  TOP1       LDA     PORTA           ;Lee el pulsador en LSB del Puerto A
0302  A4 01     [2]             AND     #$01            ;Prueba el bit-0
0304  27 FA     [3]             BEQ     TOP1            ;Lazo hasta bit-0 = 1
0306  01 00 FD  [5]  TOP2       BRCLR   0,PORTA,TOP2    ;Lazo desde aquí hasta cerrar el interruptor
0309  B6 00     [3]  TOP3       LDA     PORTA           ;Lee el pulsador en LSB del Puerto A
030B  44        [3]             LSRA                    ;Desplaza el Bit-0 para acarreo
030C  24 FB     [3]             BCC     TOP3            ;Lazo hasta pulsador cerrado
030E  A6 01     [2]             LDA     #$01            ;1 en LSB
0310  B5 00     [3]  TOP4       BIT     PORTA           ;Prueba el pulsador para bit-0
0312  27 FC     [3]             BEQ     TOP4            ;Lazo hasta interruptor cerrado
```

Figura 36. Cuatro maneras de verificar un interruptor

Los números entre corchetes, son el número de ciclos que la CPU requiere para cada instrucción de la línea de programa. La secuencia TOP1 necesita seis bytes de espacio de programa y ocho ciclos. El acumulador es $01cuando el programa llega a la declaración BEQ. La secuencia TOP2 sólo necesita tres bytes y cinco ciclos y el acumulador no estará perturbado. (Ésta probablemente es la mejor secuencia en la mayoría de casos). La secuencia TOP3 toma un byte menos que la secuencia TOP1, pero también necesita un ciclo extra en la ejecución. Después de la secuencia TOP3, el acumulador todavía mantiene los otros siete bits leídos del puerto A, aunque estos se han desplazado una posición a la derecha. La última secuencia necesita seis bytes y un total de ocho ciclos, pero el propio lazo es de sólo seis ciclos. Trabajando a través de ejercicios así, se mejorará la destreza con el juego de instrucciones. Esto será muy útil cuando se necesita reducir unos bytes de un programa para encajarlo en el espacio de memoria disponible.

Desarrollo de la Aplicación

Motorola ofrece unos sistemas de desarrollo simples para la familia MC68HC705 (por ejemplo el M68ICS05JE para el MC68HC705KJ1A y MC68HC705J1A). Estos sistemas incluyen un simulador en circuito (software y placa hardware). Las placas se conectan al puerto serie (com) de un PC. Con un cable permite, al simulador en circuito, ser conectado a una aplicación para ocupar el lugar del microcontrolador que se usará en el futuro. También en la placa de desarrollo hay un zócalo especial que permite programar una versión EPROM o OTP.

Un **simulador** es un programa para PC, que ayuda al desarrollo y la depuración de un programa. Esta herramienta simula las acciones de un microcontrolador real, pero tiene algunas ventajas importantes. Por ejemplo, en un simulador se tiene el control completo sobre ‘cuando’ y ‘si debe’ la simulación de la CPU debe adelantar a la siguiente instrucción. También se puede ‘mirar a’ y ‘cambiar los registros’ o ‘posiciones de memoria’ antes de ir a la siguiente instrucción.

Los simuladores no corren a velocidad de tiempo real. Desde el PC se simulan acciones de la MCU con programas de software, la MCU necesita más tiempo para ejecutar cada instrucción, que lo habría en una MCU real. Para muchos programas, esta reducción de velocidad no es notable. Tan lento como puede ser un simulador, todavía es mucho más rápido que en términos humanos. Algunos programas generan retardos de tiempo en lazos de software (como los de la rutina DLY50 del **Listado 3. Listado Ensamblador**). Los retardos de 50 ms de

DLY50 podrían tomar diez segundos en algún PC. Para hacer correr la simulación más rápidamente, se puede reemplazar temporalmente el valor del lazo (65) con un número mucho más pequeño (por ejemplo, 2).

NOTA: Hay que recordar de volver a poner el número original antes de programar el programa terminado en la EPROM de una MCU real.

Un **simulador en circuito** es un simulador que puede conectarse a un sistema usuario en el lugar del microcontrolador. Un simulador ordinario normalmente sólo toma información de la entrada del PC y muestra las salidas y los resultados en la pantalla del PC. Un simulador en circuito va más allá de esto, para emular las interfaces de entrada y salida del microcontrolador real.

El desarrollo del programa es más fácil con un simulador que una MCU real. Es más fácil hacer cambios del programa y probarlos en el simulador, que programar cada vez un dispositivo de EPROM y probarlo. Con la MCU real, sólo se puede ver los pins de entrada/salida y no se puede detener un programa fácilmente entre las instrucciones. Pero con el simulador, se puede ejecutar una sola instrucción en un bit determinado y mirar los registros y los contenidos de la memoria en cada paso. Esto hace más fácil ver las instrucciones que no se realizan tal como se había pensado. Un simulador también puede informar si el programa intenta usar el valor de una variable antes de que se haya inicializado.

Un **emulador en circuito** es una herramienta de desarrollo en tiempo real. El emulador se ha construido alrededor de un MCU real, para que pueda ejecutar instrucciones del programa exactamente cuando ellas se ejecutaran en la aplicación final. Un emulador tiene memoria RAM donde se localizará la memoria ROM o EPROM en la MCU final. Esto permite cargar programas rápidamente en el emulador y cambiar estos programas durante su desarrollo.

La circuitería extra en un emulador permite poner **'puntos de paro' (breakponits)** en el programa en desarrollo. Cuando el programa alcanza una de éstas direcciones de puntos de paro, el programa en desarrollo es detenido temporalmente y un **programa monitor** de desarrollo toma el control. Este programa monitor permite leer o cambiar los registros de la CPU, las posiciones de memoria o los registros de control. Un emulador típicamente tiene menos visibilidad de las acciones internas de la MCU que un simulador, pero puede correr en tiempo real. Un emulador normalmente no puede detener los relojes internos, si es necesario detiene el control del programa de aplicación y pasa dicho control al programa monitor que dispone el emulador. Un simulador puede detener estos relojes.

Resumen

El proceso de escribir un programa empieza con una buena planificación. Para ello se puede usar un organigrama para documentar dicha planificación. Las declaraciones del código fuente (mnemónicos) se escriben para cada bloque del organigrama. Las declaraciones del código fuente pueden incluir cualquiera de las instrucciones del juego de instrucciones del microcontrolador. El siguiente paso es combinar todas las instrucciones del programa con directivas del ensamblador para conseguir un archivo fuente de texto.

Las directivas del ensamblador son declaraciones del programa a las que dan instrucciones al ensamblador en lugar de la CPU. Estas instrucciones dicen cosas al ensamblador de dónde localizar instrucciones en la memoria del microcontrolador. Las directivas del ensamblador también pueden informar al ensamblador del significado binario de una etiqueta mnemónica. Se describieron seis directivas:

- ORG — Directiva de Origen, pone la dirección de inicio para el código objeto.
- EQU — Directiva de Igual, asocia una etiqueta con un número binario o dirección.
- FCB — Directiva de Byte de Forma constante, se usa para introducir un valor de 8-bits de datos constante en un programa.
- FDB — Directiva de Byte de Forma Doble, se usa para introducir un dato de 16-bits o constantes de dirección en un programa.
- RMB — Byte de Reserva de Memoria, se usan para asignar etiquetas (perteneciendo a variables del programar) para direccionar la RAM.
- $BASE 10T — Cambia por defecto el número base a decimal.

Después de que el programa fuente se haya escrito, se procesa por un ensamblador para producir un archivo listado y un archivo objeto S-record. El listado archivo es parte de la documentación del programa. El archivo objeto S-record se puede cargar en el simulador o se puede programar en un microcontrolador.

Un lazo condicional puede producir un retardo de tiempo. El retardo depende del tiempo de ejecución de las instrucciones del lazo. Una subrutina, tal como esta rutina de retardo, se puede usar muchas veces en un programa llamándola con instrucciones JSR o BSR.

La destreza de las instrucciones, es la habilidad de resolver un problema de programación de varias maneras diferentes con secuencias diferentes de instrucciones. Ya que, para ejecutar cada secuencia la CPU toma un número diferente de bytes de programa y un número diferente de ciclos, se puede seleccionar la secuencia mejor para cada situación.

Un simulador es una herramienta de desarrollo para aplicaciones, en la que corre desde un PC y simula la conducta de un microcontrolador (aunque no a velocidad de tiempo real). Un simulador en circuito toma además la idea de simular las interfaces de E/S del microcontrolador. El simulador en circuito puede conectarse en un circuito de aplicación en el lugar del microcontrolador. Un simulador hace el desarrollo de la aplicación más fácil. Permite ejecutar instrucciones en un momento determinado. También proporciona una visión de los contenidos de los registros y de la memoria, permitiendo su cambio antes de ejecutar una nueva instrucción.

Un emulador se construye alrededor de una MCU real para que pueda correr a la velocidad de la MCU final. Los emuladores usan la RAM en lugar de ROM o EPROM para que el programa en desarrollo se pueda modificar fácilmente durante su desarrollo.

Estructura del Programa Base

Índice

Introducción

Este capítulo presenta una estructura 'software' de propósito general que puede ser usada para muchas aplicaciones con microcontrolador. La mayoría de programas se escriben las tareas como subrutinas. Estas subrutinas se organizan en bucles, para que cada una de ellas sea llamada una vez por cada paso a través de un bucle. En el inicio del bucle hay una pequeña rutina que regula el bucle para que se ejecute a intervalos regulares. Un reloj por software se mantiene como la primera tarea del bucle. Este reloj se puede usar como una entrada a otras tareas de subrutina, para decidir que rutina debe hacer en cada paso, a través del bucle mayor.

Además de la propia estructura del bucle, este capítulo describe el sistema de inicialización y los detalles de configuración 'software', para que se pueda ir directamente a las rutinas que tratan las aplicaciones específicas.

EQUATES del Sistema

Ya que es un inconveniente usar modelos de bits binarios y direcciones en instrucciones de un programa, se usan las directivas EQU para asignar nombres mnemónicos a direcciones de registros y posiciones de bit. Estos nombres se pueden usar como instrucciones del programa, en lugar de números binarios. Esto hace más fácil de leer y escribir un programa.

Cuando se usa un simulador en circuito para desarrollar un programa de aplicación, se pueden usar los nombres del código mnemotécnico durante la depuración del programa, en lugar de usar las direcciones binarias.

Registros EQU para el MC68HC705J1A

Los nombres recomendados por el fabricante para los registros y los bits de control, están incluidos en el bucle regulador del programa del **Listado 4. Estructura del Programa Base,** en este capítulo. Esto permite escribir instrucciones de un programa usando nombres que tienen sentido a los humanos, en lugar de números binarios y direcciones.

Cada registro se iguala (equates) a su dirección binaria de página directa, con un directiva EQU. Cada bit de control se define de dos maneras:

- Primero, una directiva EQU, iguala el nombre del bit a un número entre 0 y 7 correspondiente al número de bit donde se localiza cada bit en un registro de control.
- Segundo, la mayoría de bits de control son igualados a un modelo de bit binario, tal como 0010 0000 ($20), qué se puede usar como una máscara de bit para identificar la posición del bit en un registro.

Puesto que no se pueden poner dos EQU con el mismo nombre a dos valores binarios diferentes, el segundo EQU se usa un periodo después del nombre de bit. Para conseguir un nombre de bit, el número de bit (7-0), usa el nombre; para conseguir una máscara que indica la posición del bit, usa el nombre seguido por un

periodo. Esta convención se usa en el bucle regulador, pero necesariamente no es una norma recomendada por Motorola o por compañías del ensamblador.

En el juego de instrucciones del MC68HC05, las instrucciones de manipulación de bit son de la forma siguiente:

```
xxxx    14  08          -----           BSET bit#,dd    ; Pone a 1 el bit en la posición dd
```

El Bit# es un número entre 0 y 7, que identifica el bit dentro del registro *dd* de la posición que será cambiada o probada.

En otros casos, se quiere construir una máscara con varios bits y entonces escribir este valor compuesto a una posición del registro. Por ejemplo, suponiendo que se quieren poner los bits RTIFR, RTIE y RT1 en el registro TSCR.

Se pueden usar estas instrucciones:

```
xxxx    A6 16       LDA # {RTIFR.+RTIE.+RT1.}   ; Forma de la máscara
xxxx    B7 08       STA TSCR                    ; Escribe la máscara al registro TSCR
```

El símbolo # significa el modo de direccionamiento inmediato. La expresión (RTIFR.+RTIE.+RT1.) es una OR Booleana de tres máscaras de posición de bit. El ensamblador evalúa la expresión Booleana durante el ensamblaje del programa y substituye la respuesta (un solo valor binario de 8 bits) en el programa ensamblado. Estas declaraciones de programa producirían exactamente el mismo resultado, pero no son fáciles de leer.

```
xxxx    A6 16       LDA #%00010110      ;Forma de la máscara
xxxx    B7 08       STA $08             ;Escribe la máscara al registro TSCR
```

Aplicación de los EQU del Sistema

Normalmente, alguna aplicación específica de las directivas EQU estará en un programa para definir las señales conectadas a los pins de E/S. Estas directivas EQU, se deben poner después de las directivas de las MCU normales y antes del inicio del programa principal. Este sistema tiene un interruptor conectado al bit-0 del puerto A y un LED conectado al bit-7 del puerto A, estas conexiones se han definido con directivas EQU.

El interruptor no se usa en la estructura del programa del bucle regulador del **Listado 4. Estructura del Programa Base**, pero no le hace ningún daño incluir las directivas EQU relacionadas. Las directivas EQU no generan ningún código objeto que ocupe espacio de memoria del sistema microcontrolador.

Preparación de los Vectores

En todos los programas se deben preparar los vectores de 'reset' y de interrupción. Los vectores especifican las direcciones donde la CPU empezará a procesar las instrucciones cuando ocurra un 'reset' o una interrupción. El 'reset' y cada fuente de interrupción esperan encontrar su vector asociado en un par de posiciones de memoria específicas. Por ejemplo, el vector de 'reset' es una de las dos posiciones más altas de la memoria ($07FE y $07FF en el MC68HC705J1A). Si no se ponen valores en estas posiciones, la CPU tomará cualquier valor binario encontrado allí y los tratará como si fueran una dirección de 2 bytes que estuviera allí guardada.

Vector de Reset

La manera usual de definir un vector es con una directiva FDB.

```
07FE    03 00       RESETV      FDB     START       ; Inicio del programa con un 'reset'
```

Durante el ensamblaje, el ensamblador evalúa la etiqueta START en una dirección de 2 bytes y guarda esta dirección en las dos siguientes posiciones de memoria disponible del programa. Las columnas de la izquierda de la línea del listado muestran que la dirección $0300 se guardó en $07FE y $07FF ($03 en $07FE y $00 en $07FF).

En esta línea de programa, RESETV es una etiqueta opcional. Aunque no se usa para referencia a través de otras declaraciones en este programa en particular, se incluyó para identificar esta línea de directiva FDB como la declaración que define el 'vector de reset'.

El 'vector de reset' fue puesto para apuntar a la etiqueta START. El simulador en circuito que ofrece Motorola, como herramienta de desarrollo económica, utiliza esta información para preparar la pantalla del simulador. Cuando un programa es cargado en el simulador, el simulador busca la dirección en el 'vector de reset' del programa cargado. Si encuentra una dirección, el simulador selecciona la instrucción del programa y la muestra en la pantalla, en la ventana del programa fuente del simulador. El simulador también se pone en esta dirección. Si hay no 'vector de reset', el simulador muestra un mensaje de advertencia y dice que el 'vector de reset' no fue inicializado. Todavía se podría poner a punto el programa, pero no trabajaría si se ha programado en una EPROM de una MCU, porque el programa no empezaría hasta que hubiera un 'reset'.

Interrupciones sin usar

Los vectores de interrupción se pueden definir como fueron definidos los 'vectores de reset' (con un directiva FDB). En el programa de bucle regulador, la interrupción del temporizador se usa para las interrupciones de tiempo real (RTI). La interrupción externa y la interrupción por software RTI no se usan.

Es una buena idea preparar los vectores de interrupción que no se van a usar, sólo en el caso de que una de estas interrupciones se soliciten inesperadamente. Esto no quiere decir que pueden ocurrir interrupciones inesperadas en el funcionamiento de un microcontrolador. Más bien, dice que cuando un programador está empezando por primera vez, los errores de programación pueden producir fuentes de interrupción imprevistas que se habilitan y se activan.

Este listado muestra cómo fueron preparados unos vectores de interrupción y de reset, en la estructura del programa del bucle regulador.

```
        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * RTIF Rutina de Servicio de Interrupción
        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0345    3A E0           RTICNT      DEC     RTIFs       ;En cada RTIF
  "      "   "            "          "       "                 "
  "      "   "            "          "       "                 "
0351    80              AnRTI       RTI                 ;Retorno de la interrupción RTIF
0351                    UNUSED      EQU     AnRTI       ;Usa RTI en AnRTI para interrupciones
                                                        ;sin usar para simplemente volver
        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * Vectores de interrupción y de reset
        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
07F8                                ORG     $07F8       ;Inicio del área de vectores
07F8    03 45           TIMVEC      FDB     RTICNT      ;Cuenta  3/TIC RTIFs
07FA    03 51           IRQVEC      FDB     UNUSED      ;Cambia si usa el vector
07FC    03 51           SWIVEC      FDB     UNUSED      ;Cambia si usa el vector
07FE    03 00           RESETV      FDB     START       ;Inicio del programa con un 'reset'
```

Las primeras líneas en este listado parcial, muestra las primeras y las últimas líneas de la rutina del servicio de interrupción del temporizador.

La siguiente línea, muestra la instrucción (RTI), retorno de la interrupción, con la etiqueta AnRTI.

```
0351    80              AnRTI       RTI                 ;Retorno de la interrupción RTIF
```

La siguiente línea iguala la etiqueta UNUSED a la dirección de la instrucción RTI en AnRTI. Más abajo del listado, los vectores de interrupción sin usar por interrupciones externas e interrupciones SWI son puestas para apuntar a esta instrucción RTI. Durante el ensamblaje, el ensamblador encuentra la etiqueta UNUSED y se debe encontrar igual a AnRTI para que sea a su vez igual a la dirección binaria de la instrucción RTI ($0351).

Si se encuentra una interrupción SWI inesperadamente, la CPU guardará los registros de la CPU en la pila (RAM temporal) y cargará el contador de programa con la dirección $0351 del vector SWI. Entonces, la CPU cargará la instrucción RTI de la dirección $0351. La instrucción RTI dirá a la CPU que recupere los registros de la

CPU guardados en la pila (incluso el contador de programa). El valor recuperado del contador de programa determina lo que hará a continuación la CPU.

Una manera diferente de responder a las interrupciones inesperadas, sería restablecer el indicador de pila (con una instrucción RSP) y entonces saltar a la misma dirección como si hubiera ocurrido un 'reset'. Esto quiere decir que si ocurre una interrupción inesperada, puede haber otro serio problema. Restableciendo el puntero de pila y volviéndolo a iniciar todo, es la forma de corregir cualquier cosa causada por una interrupción inesperada.

Mientras se depura un programa en un simulador, hay otra manera posible de manejar las interrupciones sin uso:

```
  "      "  "      "          "      "                "
0351           BADINT      BRA    BADINT     ;Bucle Infinito hasta aquí
  "      "  "      "          "      "                "
  "      "  "      "          "      "                "
07FA   03 51   VECTOR      FDB    BADINT     ; Atiende una interrupción inesperada
  "      "  "      "          "      "                "
```

En este esquema, una interrupción inesperada causará a la CPU que vaya al vector a BADINT. La instrucción a BADINT es un retorno a un bucle infinito a BADINT, para que el sistema se quede colgado en ese bucle. Se puede detener el simulador y se pueden verificar los valores de los registros de la CPU en la pila para ver lo que el programa estaba haciendo cuando se consiguió la interrupción inesperada.

Variables de la RAM

Las variables del programa cambian de valor durante el curso de la ejecución de un programa. Estos valores no se pueden especificar antes de escribir el programa y programar la MCU. La CPU debe usar instrucciones del programa para inicializar y modificar estos valores.

Cuando se escribe un programa, se reserva un espacio para las variables en la memoria RAM, usando la directiva de reserva de byte(s) de memoria (RMB).

Primero, se pone una directiva origen (ORG) para poner el contador de posición del ensamblador a la dirección de inicio de la RAM ($00C0 en el MC68HC705J1A). Cada variable o grupo de variables se pondrá con una directiva RMB. La línea RMB se identifica por el nombre de la variable. El ensamblador asigna el nombre (etiqueta) a la siguiente dirección disponible. Después de asignar cada nueva variable o grupo de variables, el contador de posición avanza para apuntar a la siguiente posición de memoria libre.

Como el programa mostrado en el **Listado 4. Estructura del Programa Base**, algunos programadores saben que es una buena práctica poner a 0 todas las posiciones de la RAM, como uno de los primeros pasos de inicialización después de cualquier 'reset'.

Mientras se depura un sistema, es útil tener un juego conocido de condiciones de arranque. Si la RAM se pone completamente a 0 al empezar un programa, es fácil decir si se ha escrito cualquier posición.

Bucle Base

El Bucle Base es una estructura de software de propósito general, que es conveniente para una amplia variedad de aplicaciones. La idea principal es romper la aplicación global en una serie de tareas como, guardar la huella de tiempo, la lectura de las entradas del sistema y actualización de las salidas del sistema. Cada tarea se escribe como una subrutina. Se construye un bucle principal fuera de las instrucciones de salto a subrutina (JSR) para cada tarea. La primera parte del bucle es un contador de secuencias software. Cuando activa el contador de secuencias, la lista de tareas de la subrutina se ejecutan una vez y una instrucción de bifurcación toma el inicio del bucle para esperar el siguiente inicio de secuencia.

La Figura 37 muestra un organigrama del bucle regulador principal. El bloque de inicio es un bucle que espera para el inicio de secuencia (cada 100 milisegundos). Los siguientes bloques tienen que ver con el mantenimiento del contador TIC. El programa **Listado 4. Estructura del Programa Base,** tiene dos simples tareas principales, TIME y BLINK. Se podría quitar una de las dos rutinas o ambas y sustituirse por tareas propias. La única limitación en el número de tareas principales, es que deben acabar bastante rápidas para no perder el inicio de secuencia. El último bloque del organigrama es una bifurcación de retorno al inicio del bucle para esperar al siguiente inicio de secuencia.

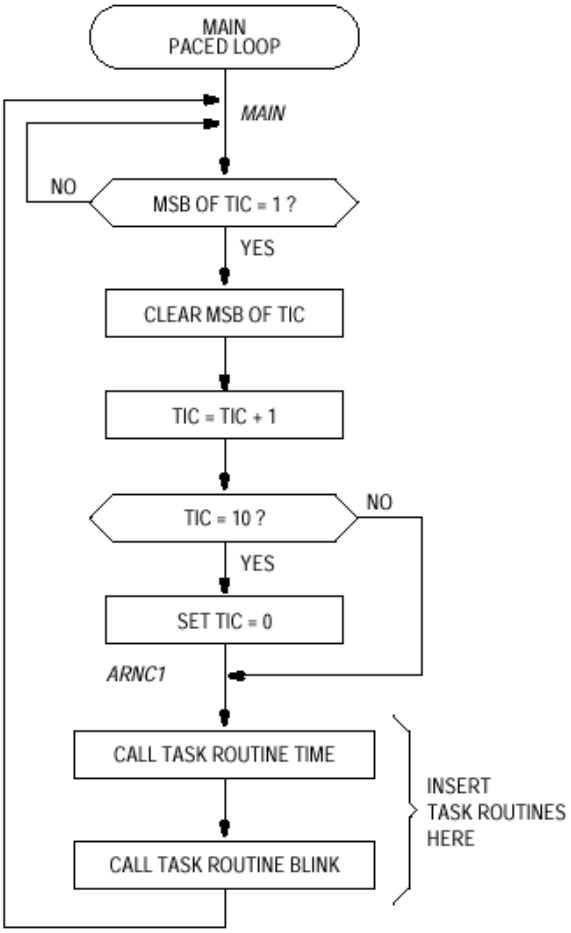


Figura 37. Organigrama del Programa Base.

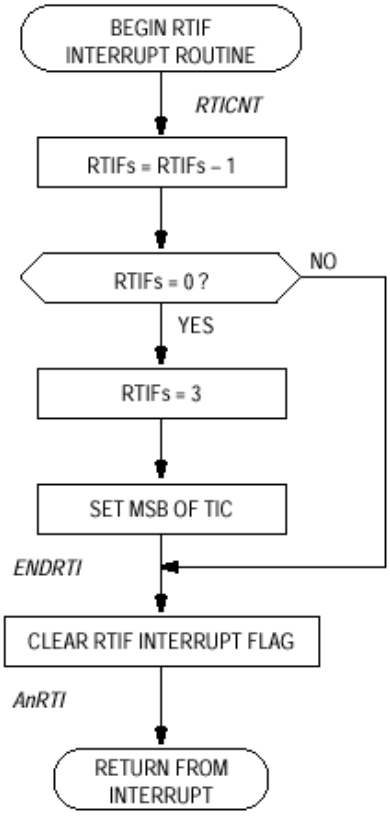


Figura 38. Organigrama de la Rutina de Servicio RTI.

Bucle Secuenciador

En el programa bucle regulador **Listado 4. Estructura del Programa Base**, el controlador de secuencia está basado en la interrupción en tiempo real interna (RTI). Esta RTI se pone para generar una interrupción a la CPU cada 32.8 milisegundos. El organigrama de la Figura 38 muestra lo que pasa en cada interrupción RTI. Esta actividad de la interrupción puede ser considerada como si estuviera teniendo lugar asíncronamente, con respecto al programa principal. El bit más significativo de la variable TIC se usa como un 'flag' para decir al programa principal cuando este tiempo se ha de incrementar TIC y ejecutar un paso a través del Bucle Base.

La variable de la RAM RTIFs se usa para contar tres interrupciones de tiempo real antes de establecer el MSB de TIC. El programa principal estará observando TIC para ver cuando el MSB se pone a 1. Cada 32.8 ms el 'flag' de RTIF se pondrá a 1 y se activará una petición de interrupción del temporizador. Uno de los deberes de una rutina de servicio de interrupción es poner a 0 el 'flag' que causó la interrupción antes de volver de la interrupción. Si RTIF no se pone a 0 antes del retorno, se generará inmediatamente una nueva petición de interrupción, en lugar de esperar el disparo cada 32.8 ms.

Bucle del Reloj del Sistema

La variable TIC es el reloj más básico para el controlador de secuencia. TIC cuenta de 0 a 10. Cuando TIC se incrementa de 9 a 10, el programa reconoce esto y poniendo TIC a 0. Excepto dentro del propio controlador de secuencia, TIC aparece para contar de 0 a 9. TIC es igual a 0 en cada décimo del inicio de secuencia.

La primera tarea de la subrutina del bucle principal se llama TIME. Esta rutina mantiene un reloj más lento llamado TOC. TOC se incrementa cada vez que se ejecuta el bucle regulador y TIC se pone a 0 (cada décimo paso a través del Bucle Base). TOC se pone como un contador software que cuenta de 0 a 59. Las rutinas

de las tareas restantes, después de TIME pueden usar los valores actuales de TIC y TOC para decidir lo que se necesita hacer en este paso, a través del Bucle Base.

En el **Listado 4. Estructura del Programa Base** la secuencia está codificada a la RTI, no pasa de ser un submúltiplo entero de 1 segundo. Los tres periodos de RTI son iguales a 98.4 milisegundos. Esto es cercano a 0.1 segundo, pero no lo suficiente preciso para ser usado como un reloj de pulsera. Se puede conseguir un tiempo real exacto si se modifica el programa del Bucle Base para usar una fuente de disparo diferente como **cruces por cero** de la línea de red (corriente alterna) de 60 Hz. Aunque la línea de red no es tan exacta como la de un cristal en periodo cortos de tiempo, es muy exacto en largos periodos de tiempo. La mayoría de los relojes conectados a la red lo utilizan como base de tiempo.

Sus Programas

Las tareas de las subrutinas tienen pocas restricciones. Cada tarea de la subrutina debe hacer todo lo que se necesita hacer, tan rápidamente como pueda y entonces ejecutar un retorno de subrutina (RTS). El tiempo total necesitado para ejecutar un paso a través de todas las tareas de la subrutina debe ser menor que dos inicios de secuencia, (posteriormente se explicará esto en mayor detalle). El punto importante es que una tarea de la subrutina no debe esperar que ocurran algunos eventos externos, como un interruptor al ser apretado. Esto provocaría retardos en la secuencia del Bucle Base.

El Bucle Base se puede preparar para los rebotes del interruptor automáticamente. Es conocido que cuando se pulsan los interruptores y se sueltan hacen rebotes. No es nada raro que cuando se pulsa un interruptor, haga rebotes de una duración de 50 ms o más. Un microcontrolador puede ejecutar las instrucciones mucho más rápidas que una sola pulsación de un pulsador a menos que se tomen los pasos para contar los rebotes del pulsador. Hay métodos hardware para eliminar los rebotes del interruptor, pero requiere más componentes y aumento del costo del producto.

También se puede usar el software para eliminar los rebotes de un interruptor, como el programa ejemplo de la Figura 31. El **diagrama de flujo** usa un retardo por software muy simple para eliminar los rebotes de un interruptor, pero esta rutina no se debe usar directamente en la estructura del bucle regulador porque toma demasiado tiempo. En un Bucle Base, se pueden eliminar los rebotes de un interruptor leyendo los pasos consecutivos a través del Bucle Base. La primera vez que se ve el interruptor pulsado, se puede escribir un valor especial a una variable para indicar que el interruptor fue pulsado provisionalmente. (Este interruptor todavía no se debería considerar como pulsado). En el siguiente paso a través del Bucle Base, tampoco se podría marcar que el interruptor se ha pulsado o se ha dejado de pulsar, con una marca para indicar que fue una falsa detección. De igual manera, cuando el interruptor se suelta, se puede marcar como una tentativa de haberse soltado y en el siguiente paso por la marca es como si se hubiera soltado realmente.

Consideraciones de tiempo

A ser posible, se deben terminar todas las tareas de las subrutinas en el bucle regulador antes de que llegue el siguiente inicio de secuencia. Si un simple paso a través del bucle, necesita mucho más tiempo que los periodos de inicio de secuencia, el 'flag' que indica que es el momento de empezar el siguiente paso a través del bucle principal, se pondrá cuando se vuelva al inicio del bucle. No pasa nada malo a menos que se consiga detrás de hasta ahora que un nuevo inicio de secuencia venga antes de que el anterior sea reconocido. El Bucle Base permanece válido a menos que cualquiera de los dos pasos consecutivos necesite más de dos periodos de inicio de secuencia.

Un poco de planificación puede asegurar que ninguno de los dos pasos consecutivos a través del bucle necesite mucho más tiempo que dos periodos de control de secuencia. Especialmente las largas tareas de subrutinas se pueden fijar para ejecutar durante un particular paso del bucle regulador cuando otra actividad muy pequeña sea fijada. Se puede usar una simple inspección de una de las variables de tiempo como TIC o TOC para decidir si o no realizar una rutina particularmente lenta. Si se tuvieran que hacer varias veces por segundo, un bucle podría fijar el paso de TIC = 0 y otro bucle podría fijar el paso de TIC = 2 y así sucesivamente.

Consideraciones de la Pila (Stack)

Los Microcontroladores pequeños como el MC68HC705J1A, tienen la RAM muy pequeña para la pila y para las variables del programa. Las interrupciones utilizan cinco bytes de RAM de la pila y cada llamada a subrutina utiliza dos bytes en la pila. Si una subrutina llama a otra subrutina y es solicitada una interrupción antes de terminar la segunda subrutina, la pila habrá utilizado 2 + 2 + 5 = 9 bytes de RAM, de los 64 bytes disponibles. Si la pila ocupa mucho lugar en la RAM, hay el peligro de que las variables de la RAM puedan escribirse encima de los datos de la pila. Para evitar estos problemas, se debe calcular el peor de los casos en que

se puede conseguir la ocupación de la pila. En el MC68HC705J1A, la suma de todo el sistema de variables, más el peor de los casos de ocupación de la pila, debe ser menor o igual a las 64 posiciones disponibles de la RAM.

Afortunadamente, una interrupción causará la máscara de interrupción del bit I en el registro de código de condición, para ser puesto en contestación de cualquier interrupción. Este bloque de interrupciones adicionales hasta el bit I se pondrá a 0 (normalmente en el retorno de la interrupción).

Estructura de una Aplicación Preparada

El programa del Bucle Base del **Listado 4. Estructura del Programa Base,** se puede usar como base para las aplicaciones. Esta estructura proporciona:

- Declaraciones EQU para todos los registros del MC68HC705J1A y nombres de bit
- Declaraciones EQU de aplicaciones específicas
- Sección de variables del programa
- Sección de inicialización (START)
- Controlador de secuencia para el bucle principal basado en la interrupción RTI
- Llamada a tareas de subrutinas
- Dos ejemplos simples de tareas de subrutinas (TIME y BLINK)
- Una rutina de servicio de interrupción para las interrupciones RTIF
- Sección de definición de vectores

El inicio de secuencia en este programa de Bucle Base, activa un paso a través del bucle principal una vez cada 100 milisegundos (realmente 98.4 ms). Esto se puede cambiar fácilmente a otro número de interrupciones de tiempo real y se puede cambiar el valor RTI. Para aplicaciones que se necesita un reloj (RTC), los inicios de secuencias se pueden modificar para trabajar desde interrupciones generadas por cruce por cero de la línea de red alterna.

Adicionalmente se deben agregar directivas RMB a la sección de variables del programa. Se pueden agregar declaraciones EQU adicionales justo sobre la sección de variables del programa para añadir EQU de aplicación específica.

En el programa del **Listado 4. Estructura del Programa Base**, el bucle regulador tiene sólo dos subrutinas de tareas simples (TIME y BLINK). La tarea de TIME proporciona un contador de 0 a 59 (TOC) qué podría ser útil para medir o generar periodos de tiempo más largos. La tarea de BLINK es una rutina para demostrar cómo una tarea puede usar la variable de tiempo TOC para controlar una acción del sistema. En este caso, la acción es encender un LED cuando TOC es par y apagarlo cuando TOC es impar. Para usar la estructura del programa para cualquier aplicación, se debe quitar la tarea de BLINK y reemplazarla con sus tareas correspondientes.

Las rutinas de servicio RTI, sirven como ejemplo para el manejo de una interrupción e interrupciones de contador en tiempo real, para poner la velocidad del controlador de secuencias.

Listado 4. Estructura del Programa Base (pág 1 de 5)

```
$BASE 10T
*********************************************************
* EQUs para el MC68HC705J1A MCU
* Usa nombres de bit sin un punto en BSET..BRCLR
* Usa nombre de bit precedido por un punto en expresiones como
* #.ELAT+.EPGM para formar una máscara de bit
*********************************************************
PORTA       EQU   $00         ;E/S puerto A
PA7         EQU   7           ;Bit #7 del puerto A
PA6         EQU   6           ;Bit #6 del puerto A
PA5         EQU   5           ;Bit #5 del puerto A
PA4         EQU   4           ;Bit #4 del puerto A
PA3         EQU   3           ;Bit #3 del puerto A
PA2         EQU   2           ;Bit #2 del puerto A
PA1         EQU   1           ;Bit #1 del puerto A
PA0         EQU   0           ;Bit #0 del puerto A
PA7.        EQU   $80         ;Posición del bit PA7
PA6.        EQU   $40         ;Posición del bit PA6
PA5.        EQU   $20         ;Posición del bit PA5
PA4.        EQU   $10         ;Posición del bit PA4
PA3.        EQU   $08         ;Posición del bit PA3
PA2.        EQU   $04         ;Posición del bit PA2
PA1.        EQU   $02         ;Posición del bit PA1
PA0.        EQU   $01         ;Posición del bit PA0
PORTB       EQU   $01         ;E/S puerto B
PB5         EQU   5           ;Bit #5 del puerto B
PB4         EQU   4           ;Bit #4 del puerto B
PB3         EQU   3           ;Bit #3 del puerto B
PB2         EQU   2           ;Bit #2 del puerto B
PB1         EQU   1           ;Bit #1 del puerto B
PB0         EQU   0           ;Bit #0 del puerto B
PB5.        EQU   $20         ;Posición del bit PB5
PB4.        EQU   $10         ;Posición del bit PB4
PB3.        EQU   $08         ;Posición del bit PB3
PB2.        EQU   $04         ;Posición del bit PB2
PB1.        EQU   $02         ;Posición del bit PB1
PB0.        EQU   $01         ;Posición del bit PB0
DDRA        EQU   $04         ;Dirección del Dato para el puerto A
DDRA7       EQU   7           ;Bit #7 del puerto A DDR
DDRA6       EQU   6           ;Bit #6 del puerto A DDR
DDRA5       EQU   5           ;Bit #5 del puerto A DDR
DDRA4       EQU   4           ;Bit #4 del puerto A DDR
DDRA3       EQU   3           ;Bit #3 del puerto A DDR
DDRA2       EQU   2           ;Bit #2 del puerto A DDR
DDRA1       EQU   1           ;Bit #1 del puerto A DDR
DDRA0       EQU   0           ;Bit #0 del puerto A DDR
DDRA7.      EQU   $80         ;Posición del bit DDRA7
DDRA6.      EQU   $40         ;Posición del bit DDRA6
DDRA5.      EQU   $20         ;Posición del bit DDRA5
DDRA4.      EQU   $10         ;Posición del bit DDRA4
DDRA3.      EQU   $08         ;Posición del bit DDRA3
DDRA2.      EQU   $04         ;Posición del bit DDRA2
DDRA1.      EQU   $02         ;Posición del bit DDRA1
DDRA0.      EQU   $01         ;Posición del bit DDRA0
DDRB        EQU   $05         ;Dirección Dato para puerto B
DDRB5       EQU   5           ;Bit #5 del puerto B DDR
DDRB4       EQU   4           ;Bit #4 del puerto B DDR
DDRB3       EQU   3           ;Bit #3 del puerto B DDR
DDRB2       EQU   2           ;Bit #2 del puerto B DDR
DDRB1       EQU   1           ;Bit #1 del puerto B DDR
```

Listado 4. Estructura del Programa Bucle Regulador (pág 2 de 5)

```
DDRB0       EQU   0           ;Bit #0 del puerto B DDR
DDRB5.      EQU   $20         ;Posición del bit DDRB5
DDRB4.      EQU   $10         ;Posición del bit DDRB4
DDRB3.      EQU   $08         ;Posición del bit DDRB3
DDRB2.      EQU   $04         ;Posición del bit DDRB2
DDRB1.      EQU   $02         ;Posición del bit DDRB1
DDRB0.      EQU   $01         ;Posición del bit DDRB0
TSCR        EQU   $08         ;Registro de estado del Timer
                              ;y del registro de control
TOF         EQU   7           ;Flag de desbordamiento del Timer
RTIF        EQU   6           ;Flag de la interrupción en tiempo real
TOIE        EQU   5           ;Habilita interrupción TOF
RTIE        EQU   4           ;Habilita interrupción RTI
TOFR        EQU   3           ;Reset del Flag TOF
RTIFR       EQU   2           ;Reset del Flag RTIF
RT1         EQU   1           ;Selecciona la velocidad RTI bit 1
RT0         EQU   0           ;Selecciona la velocidad RTI bit 0
TOF.        EQU   $80         ;Posición del bit TOF
RTIF.       EQU   $40         ;Posición del bit RTIF
TOIE.       EQU   $20         ;Posición del bit TOIE
RTIE.       EQU   $10         ;Posición del bit RTIE
TOFR.       EQU   $08         ;Posición del bit TOFR
RTIFR.      EQU   $04         ;Posición del bit RTIFR
RT1.        EQU   $02         ;Posición del bit RT1
RT0.        EQU   $01         ;Posición del bit RT0
TCR         EQU   $09         ;Registro del contador Timer
ISCR        EQU   $0A         ;registro del estado y de control del IRQ
IRQE        EQU   7           ;flanco/nivel del flanco de IRQ
IRQF        EQU   3           ;Flag de la interrupción externa
IRQR        EQU   1           ;Reset del Flag IRQF
PDRA        EQU   $10         ;Registro Pulldown para puerto A
PDIA7       EQU   7           ;Desactiva 'pulldown' para  PA7
PDIA6       EQU   6           ;Desactiva 'pulldown' para  PA6
PDIA5       EQU   5           ;Desactiva 'pulldown' para  PA5
PDIA4       EQU   4           ;Desactiva 'pulldown' para  PA4
PDIA3       EQU   3           ;Desactiva 'pulldown' para  PA3
PDIA2       EQU   2           ;Desactiva 'pulldown' para  PA2
PDIA1       EQU   1           ;Desactiva 'pulldown' para  PA1
PDIA0       EQU   0           ;Desactiva 'pulldown' para  PA0
PDIA7.      EQU   $80         ;Posición del bit PDIA7
PDIA6.      EQU   $40         ;Posición del bit PDIA6
PDIA5.      EQU   $20         ;Posición del bit PDIA5
PDIA4.      EQU   $10         ;Posición del bit PDIA4
PDIA3.      EQU   $08         ;Posición del bit PDIA3
PDIA2.      EQU   $04         ;Posición del bit PDIA2
PDIA1.      EQU   $02         ;Posición del bit PDIA1
PDIA0.      EQU   $01         ;Posición del bit PDIA0
PDRB        EQU   $11         ;Registro Pulldown para puerto B
PDIB5       EQU   5           ;Desactiva 'pulldown' para  PB5
PDIB4       EQU   4           ;Desactiva 'pulldown' para  PB4
PDIB3       EQU   3           ;Desactiva 'pulldown' para  PB3
PDIB2       EQU   2           ;Desactiva 'pulldown' para  PB2
PDIB1       EQU   1           ;Desactiva 'pulldown' para  PB1
PDIB0       EQU   0           ;Desactiva 'pulldown' para  PB0
PDIB5.      EQU   $20         ;Posición del bit PDIB5
PDIB4.      EQU   $10         ;Posición del bit PDIB4
PDIB3.      EQU   $08         ;Posición del bit PDIB3
PDIB2.      EQU   $04         ;Posición del bit PDIB2
PDIB1.      EQU   $02         ;Posición del bit PDIB1
PDIB0.      EQU   $01         ;Posición del bit PDIB0
EPROG       EQU   $18         ;Registro de programación de la EPROM
```

Listado 4. Estructura del Programa Bucle Regulador (pág 3 de 5)

```
ELAT        EQU   2           ;control latch EPROM
MPGM        EQU   1           ;control de programación del bit MOR
EPGM        EQU   0           ;control de programación de la EPROM
ELAT.       EQU   $04         ;Posición del bit ELAT
MPGM.       EQU   $02         ;Posición del bit MPGM
EPGM.       EQU   $01         ;Posición del bit EPGM
COPR        EQU   $07F0       ;Registro de reset del 'watchdog' COP
COPC        EQU   0           ;Borra el 'watchdog' COP
COPC.       EQU   $01         ;Posición del bit COPC
MOR         EQU   $07F1       ;Registro de opción de Máscara
SOSCD       EQU   7           ;Habilita un corto retardo en el oscilador
EPMSEC      EQU   6           ;Seguridad de la EPROM
OSCRES      EQU   5           ;Oscilador con resitencia en paralelo
SWAIT       EQU   4           ;instrucción en modo STOP
PDI         EQU   3           ;Desactiva 'pulldown' del Puerto
PIRQ        EQU   2           ;Habilita la IRQ en Port A
LEVEL       EQU   1           ;Sensibilidad del flanco de la IRQ
COP         EQU   0           ;Habilita 'watchdog' COP
SOSCD.      EQU   $80         ;Posición del bit SOSCD
EPMSEC.     EQU   $40         ;Posición del bit EPMSEC
OSCRES.     EQU   $20         ;Posición del bit OSCRES
SWAIT.      EQU   $10         ;Posición del bit SWAIT
PDI.        EQU   $08         ;Posición del bit PDI
PIRQ.       EQU   $04         ;Posición del bit PIRQ
LEVEL.      EQU   $02         ;Posición del bit LEVEL
COPEN.      EQU   $01         ;Posición del bit COPEN

* EQU del area de Memoria
RAMStart    EQU   $00C0       ;Inicio de la RAM interna
ROMStart    EQU   $0300       ;Inicio de la ROM interna
ROMEnd      EQU   $07CF       ;Fin de la ROM interna
Vectors     EQU   $07F8       ;Area del vector de Reset/interrupción

* EQU de la aplicación específica
LED         EQU   PA7         ;LED ON cuando PA7 = 0
LED.        EQU   PA7.        ;Posición del bit LED
SW          EQU   PA0         ;Conmuta PA0 a 1, cerrado = 1
SW.         EQU   PA0.        ;Posición del bit Switch

*******************************************************
* Pone las variables del programa aquí (usa RMBs)
*******************************************************
            ORG   $00C0       ;Inicio de la RAM del 705J1A
RTIFs       RMB   1           ;3 RTIFs/TIC (3-0)
TIC         RMB   1           ;10 TICs hace 1 TOC (10-0)
                              ;MSB=1 significa RTIFs 'rolled over'
TOC         RMB   1           ;1 TOC=10*96.24ms= aproximadamente 1 segundo

*******************************************************
* Inicio del área de programa aquí
*******************************************************
            ORG   $0300             ;Inicio de la EPROM del 705J1A
* Primero inicializa el registro de control y variables
START       CLI                     ;Pone a 0 el bit I para interrupciones
            LDA   #LED.             ;Configura y apaga el LED
            STA   PORTA             ;Apaga el LED
            STA   DDRA              ;Pone en el pin LED como salida
            LDA   #{RTIFR.+RTIE.+RT1.}
            STA   TSCR              ;para borrar y activar RTIF
                                    ;y pone RTI para 32.8 ms
```

Listado 4. Estructura del Programa Bucle Regulador (pág 4 de 5)

```
            LDA    #3                ;RTIFs cuenta 3->0
            STA    RTIFs             ;Reset del contador TOFS
            CLR    TIC               ;Valor inicial para TIC
            CLR    TOC               ;Valor inicial para TOC

*******************************************************
* MAIN - Empieza el bucle principal del programa
* El bucle se ejecuta cada 100 ms (98.4 ms)
* Necesita 100 ms para pasar a través de todas las rutinas
* de mayor tarea y entonces el tiempo se ha consumido hasta
* que MSB de TIC se pone a 1 (cada 3 RTIFs = 98.4 ms).
* A cada interrupción RTIF, se borra RTIF y RTIFs
* consigue decrementar (3-0). Cuando RTIFs = 0, MSB de
* TIC se pone a 1 y RTIFs se pone a 3.
* (3*32.8/RTIF = 98.4 ms).
*
* La variable TIC guarda la traza de períodos de 100mS
* Cuando TIC incrementa de 9 a 10 es borrado
* a 0 y TOC es incrementado.
*******************************************************
MAIN        CLRA                     ;activa el 'watchdog'
            STA    COPR              ;si está habilitado
            BRCLR 7,TIC,MAIN         ;Bucle aquí hasta flanco TIC
            LDA    TIC               ;carga el valor actual de TIC
            AND    #$0F              ;Borra MSB
            INCA                     ;TIC = TIC+1
            STA    TIC               ;Actualiza TIC
            CMP    #10               ;¿décimo TIC?
            BNE    ARNC1             ;Si no, salta al siguiente borrado
            CLR    TIC               ;Borra TIC en 10th
ARNC1       EQU    *                 ;

* Fin de sincronización a 100 ms TIC; Corre la tarea principal
* y bifurca a MAIN en 100 ms. Sync está bien en tanto tiempo
* como 2 pasos de tareas no consecutivas, que es más de 196.8 ms
            JSR    TIME              ;Actualiza TOCs
            JSR    BLINK             ;parpadea el LED

* Otra tarea principal iría aquí
            BRA    MAIN              ;vuelve al inicio para el siguiente TIC


** Fin del Bucle Principal **********************************


*******************************************************
* TIME - Actualiza TOCs
* Si TIC = 0, incrementa 0->59
* Si TIC no es = 0, salta la rutina entera
*******************************************************
TIME        EQU    *                 ;Actualiza TOCs
            TST    TIC               ;Prueba para TIC = cero
            BNE    XTIME             ;Si no; justo salir
            INC    TOC               ;TOC = TOC+1
            LDA    #60
            CMP    TOC               ;hace TOC -> 60 ?
            BNE    XTIME             ;Si no, justo salir
            CLR    TOC               ;TOCs 'rollover'
XTIME       RTS                      ;Volver desde TIME
```

Listado 4. Estructura del Programa Bucle Regulador (pág 5 de 5)

```
*******************************************************
* BLINK - Actualiza LED
* Si TOC es par, enciende el LED
* el resto apaga el LED
*******************************************************
BLINK       EQU   *                 ;Actualiza el LED
            LDA   TOC               ;Si es par, LSB será cero
            LSRA                    ;Desplaza LSB para acarreo
            BCS   LEDOFF            ;Si no, apaga el LED
            BSET  LED,PORTA         ;Enciende el LED
            BRA   XBLINK            ;Entonces salir
LEDOFF      BCLR  LED,PORTA         ;Apaga el LED
XBLINK      RTS                     ;Volver desde BLINK

*******************************************************
* Rutina de servicio de interrupción RTIF
*******************************************************
RTICNT      DEC   RTIFs             ;Cada 'On' RTIF decrementa RTIFs
            BNE   ENDTOF            ;Realizado si RTIFs no es = 0
            LDA   #3                ;RTIFs cuenta 3->0
            STA   RTIFs             ;Reset del contador TOFS
            BSET  7,TIC             ;Pone MSB como un flag a MAIN
ENDTOF      BSET  RTIFR,TSCR        ;Borra el flag RTIF
AnRTI       RTI                     ;Vuelve de la interrupción RTIF
UNUSED      EQU   AnRTI             ;Usa RTI en AnRTI si no usa
                                    ;interrupciones justo volver

*******************************************************
* Vector de Interrupción y Reset
*******************************************************
            ORG   $07F8             ;Area de Inicio del vector
TIMVEC      FDB   RTICNT            ;Cuenta RTIFs 3/TIC
IRQVEC      FDB   UNUSED            ;Cambia si el vector es usado
SWIVEC      FDB   UNUSED            ;Cambia si el vector es usado
RESETV      FDB   START             ;Empieza el program con un reset
```

Resumen

Las directivas EQU se usan para asociar una etiqueta a un valor binario. El valor binario puede ser una dirección o una constante numérica.

Hay dos maneras diferentes de igualar un bit de control, dependiendo de cómo se usará la etiqueta. Para instrucciones de 'bit set', 'bit clear' y bifurcación, se usan la EQU para asociar una etiqueta con un número entre 0 y 7. Para construir una máscara lógica, la EQU se usa para igualar una etiqueta con una máscara de bit, donde este bit se pone en la misma posición de bit que el bit de control.

El vector de reset y el de interrupción se deben inicializar para formar una directiva doble de byte (FDB). Incluso, si no se va ha usar una fuente de interrupción, es recomendable inicializar el vector en caso de que sea generada una petición inesperada.

El espacio reservado en la RAM para las variables del programa, se hace usando las directivas de reserva de bytes de memoria (RMB).

La estructura de software del Bucle Base, es una buena estructura de programación de propósito general. Una estructura de bucle, se establece con un controlador de secuencias en el inicio del bucle. La activación controlador de secuencias, hace ejecutar las otras instrucciones del bucle a intervalos de tiempo regulares, cada 100 milisegundos. Las tareas para una aplicación se escriben como subrutinas. La lista de instrucciones de salto a subrutinas (JSR) en el bucle regulador principal, provoca que se ejecute cada subrutina precisamente una vez que se activa por el controlador de secuencias.

Las rutinas en el bucle principal se deben diseñar para que la combinación de tiempo de la ejecución de todas las rutinas del bucle sea menor que el período del inicio de secuencia. Un paso individual a través del bucle puede utilizar más tiempo el período de inicio de secuencia, con tal de que el siguiente paso sea más corto. La sincronización del bucle se mantiene tanto tiempo como dos pasos no consecutivos a través del bucle principal, entonces hay que utilizar más tiempo que dos veces el periodo del controlador de secuencias.

En los microcontroladores más pequeños, el número de posiciones de RAM disponible es pequeño, por lo que es importante ser consciente de los requisitos de la pila (stack). Una interrupción requiere cinco bytes de RAM del 'stack' y una llamada a subrutina requiere dos bytes en un MC68HC05.

Periféricos Internos

Índice

Introducción

Para resolver problemas del mundo real, un microcontrolador debe tener además más de una CPU potente, un programa y recursos de memoria de datos. Además, debe contener hardware que le permita a la CPU acceder a la información del mundo exterior. Una vez que la CPU recoge la información y procese los datos, también debe poder efectuar cambios en del mundo exterior. Éstos dispositivos hardware, llamados periféricos, son la ventana al exterior de la CPU. Los periféricos internos extienden las capacidades de los microcontroladores y reducen la carga de proceso en la CPU.

La forma más básica del periférico disponible en microcontroladores, es el puerto de E/S de propósito general. El MC68HC705J1A tiene 14 pins de E/S que están colocados en dos puertos, un puerto de 8-bits y un puerto de 6-bits. Cada uno de los pins de E/S se pueden usar como una entrada o una salida. La función de cada pin se determina poniendo a 1 o a 0 los bits correspondientes de un registro de dirección de datos (DDR) durante la fase de inicialización del programa. Cada pin de salida se puede poner a 1 o a 0 usando las instrucciones de la CPU para poner a 1 o a 0 el bit correspondiente en el registro de datos del puerto. También, el estado de la lógica de cada pin de entrada puede ser leída por la CPU usando las instrucciones del programa.

Los periféricos internos proporcionan una interface con el mundo exterior de la CPU. Los periféricos aumentan las capacidades de la CPU realizando las tareas que para la CPU no son buenas. La mayoría de periféricos del microcontrolador realizan tareas o funciones muy específicas. Por ejemplo, un periférico puede ser capaz de generar una frecuencia y medir el ancho de un pulso o puede generar formas de onda.

Ya que la mayoría de periféricos no tiene inteligencia propia, requieren alguna cantidad de ayuda de la CPU. Para impedir que la CPU requiera una atención constante, los periféricos realizan a menudo sus funciones en forma de interrupciones. La CPU hace una petición de servicio de un periférico sólo cuando requiere datos adicionales para realizar su trabajo o cuando un periférico tiene una información que la CPU necesita para hacer su trabajo. Los periféricos pueden ser sumamente potentes y pueden realizar funciones complejas sin ninguna intervención de la CPU, una vez estos se han activado. Sin embargo, debido a la sensibilidad al costo de la mayoría de dispositivos de la familia MC68HC05, los periféricos usados por estos requieren una justa cantidad de intervención de la CPU.

Tipos de Periféricos

Con la excepción de los puertos de E/S de propósito general, la mayoría de periféricos realizan tareas muy específicas. Estas tareas pueden ser diversas y pueden ir desde la medida de tiempo y el cálculo, hasta la comunicación con otros microcontroladores o periféricos externos. Los párrafos siguientes contienen una descripción general de algunos tipos de periféricos encontrados en los microcontroladores de la familia MC68HC05.

Temporizadores

Aunque existe una amplia variedad de temporizadores en la familia MC68HC05, sus funciones básicas están relacionadas con la medida o generación de eventos basados en el tiempo. Los temporizadores normalmente miden tiempo relativo al reloj interno del microcontrolador, aunque algunos pueden tener una entrada de reloj externa. Las capacidades de los temporizadores de cada dispositivo de la familia MC68HC05 varían sensiblemente. Por ejemplo, el módulo temporizador más sofisticado de la familia es el MC68HC05Bx que puede generar dos salidas PWM simultáneamente, mide el ancho de pulso de dos señales externas y genera dos salidas de trenes de pulso adicionales. En comparación, el temporizador más simple, sólo presente en el MC68HC05Jx y MC68HC05Kx, genera dos interrupciones periódicas; una a proporción fija y una a proporción seleccionable.

Los módulos temporizadores mucho más sofisticados que existen en Motorola están en los procesadores más potentes. Por ejemplo, el MC68332 y MC68HC16Y1 que contienen una unidad de proceso de tiempo (TPU) que es un procesador programable de tiempo, con microcódigo, con su propia ALU (unidad aritmética lógica). El modulo TPU fue diseñado sobre todo para el control de un motor de explosión y puede hacer funcionar una máquina a un estado constante sin la intervención de la CPU.

Puertos Serie

Algunos miembros de la familia MC68HC05 contienen periféricos que permiten a la CPU enviar bits consecutivamente a otros dispositivos externos. Usando una estructura de bit serie en lugar de un formato de bit paralelo, requiere menos pins de E/S para realizar la función de comunicación.

Existen dos tipos básicos de puertos serie en la familia MC68HC05:

- **SCI**: Interface de Comunicación Serie
- **SPI**: Interface Serie de Periféricos

El puerto SCI es un puerto transmisor/receptor asíncrono universal (UART), es decir que se comunica de forma asíncrona con otros dispositivos. Este tipo de puerto serie requiere la interface hardware más simple. Sólo se requieren dos pins para la transmisión de datos bidireccional. Los datos se transmiten de la MCU por un pin y se reciben por otro pin. Cada conjunto de datos transmitidos o recibidos por el SCI tiene un bit de inicio (start), varios bits de datos y un bit de parada (stop). El bit de inicio (start) y el bit de parada (stop), se usan para sincronizar los dos dispositivos que se están comunicando. Este tipo de interface serie se usa a menudo cuando un microcontrolador se debe comunicar a distancias moderadas (aproximadamente 12m). Con convertidores de nivel de tensión (tipo ADM232) conectados al los pins de transmisión y de recepción, el SCI se puede utilizar para comunicarse con ordenadores personales o con otros sistemas con microcontrolador con el estándar RS232.

Como su nombre implica, el puerto SPI se usa para comunicar principalmente con periféricos externos baratos (a poca distancia). El SPI comunica síncrona con otros dispositivos, transfiriendo los datos en forma bidireccional y requiere tres pins de la MCU. Además de un pin para transmitir y recibir datos, un tercer pin proporciona el reloj de la sincronización para los dispositivos a comunicar. Este tipo de interface serie, normalmente se usa para comunicar con dispositivos periféricos en la misma placa de circuito impreso.

Hay muchos dispositivos periféricos SPI disponibles por muchos fabricantes. Convertidores A/D y D/A, controladores de 'displays', EEPROMs y registros de desplazamiento, son algunos ejemplos de periféricos SPI disponibles.

Convertidor Analógico a Digital

Tal como se mencionó en el capitulo **¿Qué es un Microcontrolador?**, existen muchas señales en el mundo real, que no son directamente compatibles con los pins de E/S de la MCU. De hecho, hay muchas señales analógicas que están continuamente variando y no pueden ser continuamente traducidas a una lógica 1 o 0, que los microcontrolador pueden usar. Algunos miembros de la familia MC68HC05 incluyen un convertidor analógico a digital (A/D) que puede ser usado para convertir el nivel de voltaje analógico de las señales, en un número binario que la MCU puede usar.

Convertidor Digital a Analógico

Un convertidor digital a analógico (D/A) realiza simplemente la función opuesta de un convertidor A/D. Le permite a la MCU convertir un número digital en un voltaje analógico proporcional o a una corriente que pueden usarse para controlar diversos dispositivos de salida en un sistema. Después en este capítulo, se desarrolla una aplicación que muestra cómo un convertidor A/D puede llevarse a cabo, usando un temporizador interno y un programa software.

EEPROM

Puesto que una EEPROM es un tipo de memoria, la mayoría de usuarios no lo considera un periférico. Sin embargo, los contenidos de una EEPROM pueden modificarse con un programa y es una memoria no volátil, que es eléctricamente borrable y es diferente de una RAM, ROM o EPROM. Varios miembros de la familia MC68HC05 contienen una memoria EEPROM interna. Como se mencionó previamente, la EEPROM también se puede agregar a un sistema como un periférico SPI externo.

Control de Periféricos

El control y el estado de la información para periféricos aparecen en la CPU como bits de datos en una posición de memoria determinada. Usando este tipo de arreglo, los registros de control y de estado de los periféricos, es conocido como mapa de memoria de E/S. Es una gran ventaja el tener los periféricos que aparezcan como una posición de memoria. Cualquier instrucción de la CPU que puede operar en una posición de memoria, se puede usar para controlar o verificar el estado de un periférico. Este tipo de arquitectura de E/S es especialmente ventajoso con la familia MC68HC05 debido a las instrucciones de manipulación de bit de la CPU. Este grupo de las instrucciones le da a un programador, la habilidad de poner a 1, a 0 individualmente o la prueba del estado de cualquier bit en los registros de control de periféricos entre las direcciones $0000-$00FF.

Dependiendo del tipo y complejidad de un periférico, su control asociado y los registros de estado pueden ocupar una o varias posiciones en el mapa de memoria del microcontrolador. Por ejemplo, un puerto de E/S de propósito general ocupa dos posiciones de memoria en el mapa de memoria de un microcontrolador. Una posición de byte, llamada registro de dirección de datos (DDR), se usa para controlar la función de cada pin de E/S. La otra posición de byte, registro de datos del puerto, se usa para leer el estado de los pins de entrada o imponer un nivel lógico 1 en un pin de salida. Un periférico complejo, como el temporizador en el MC68HC705C8, ocupa 10 posiciones de byte, en el mapa de memoria de la MCU.

Mirando detalladamente el temporizador del MC68HC705J1A, se puede ver que este temporizador multifunción de 15 etapas, es muy simple comparado con otros sistemas de temporizador, pero puede realizar unas sofisticadas funciones de temporizador. Se describe un ejemplo completo, mostrando cómo este sistema temporizador se puede usar para generar una exacta baja frecuencia de PWM.

Temporizador del MC68HC705J1A

En la Figura 39 se muestra un diagrama de bloques del temporizador multifunción de 15 etapas del MC68HC705J1A. El temporizador consiste en tres secciones conectadas, cada una de estas secciones realiza funciones de temporizador separadamente.

La cadena de temporización empieza con el reloj del bus interno del microcontrolador, el E-clock. El E-clock viene de dividir la frecuencia del cristal por dos. A continuación el E-clock se divide por cuatro. A su vez, la salida va a un contador de 8-bits. El valor de este contador, se puede leer por la CPU en cualquier momento en la posición de memoria $09, en el registro del contador del temporizador (TCR). El valor del contador no puede ser alterado por la CPU. Esto puede parecer como un temporizador simple; sin embargo, es útil en muchas aplicaciones. Cuando el contador de 8-bits se desborda de $FF a $00, se pone a 1 el bit de estado del 'flag' de desbordamiento del temporizador (TOF) del control del temporizador y del registro de estado (TCSR). El estado de este 'flag' de estado, se puede probar en cualquier momento, por cualquiera de las diferentes instrucciones de la CPU.

Opcionalmente, si la habilita la interrupción por desbordamiento del temporizador (TOIE), el bit de control del temporizador y del registro de estado se pone a 1, el desbordamiento del contador de 8 bits, generará una interrupción en la CPU. Por consiguiente, la función de desbordamiento del temporizador permite generar una interrupción muy potente. El desbordamiento del temporizador se produce cada 1024 ciclos de E-clock (primero dividido por cuatro seguido por un contador de 8-bits, divido por un 256).

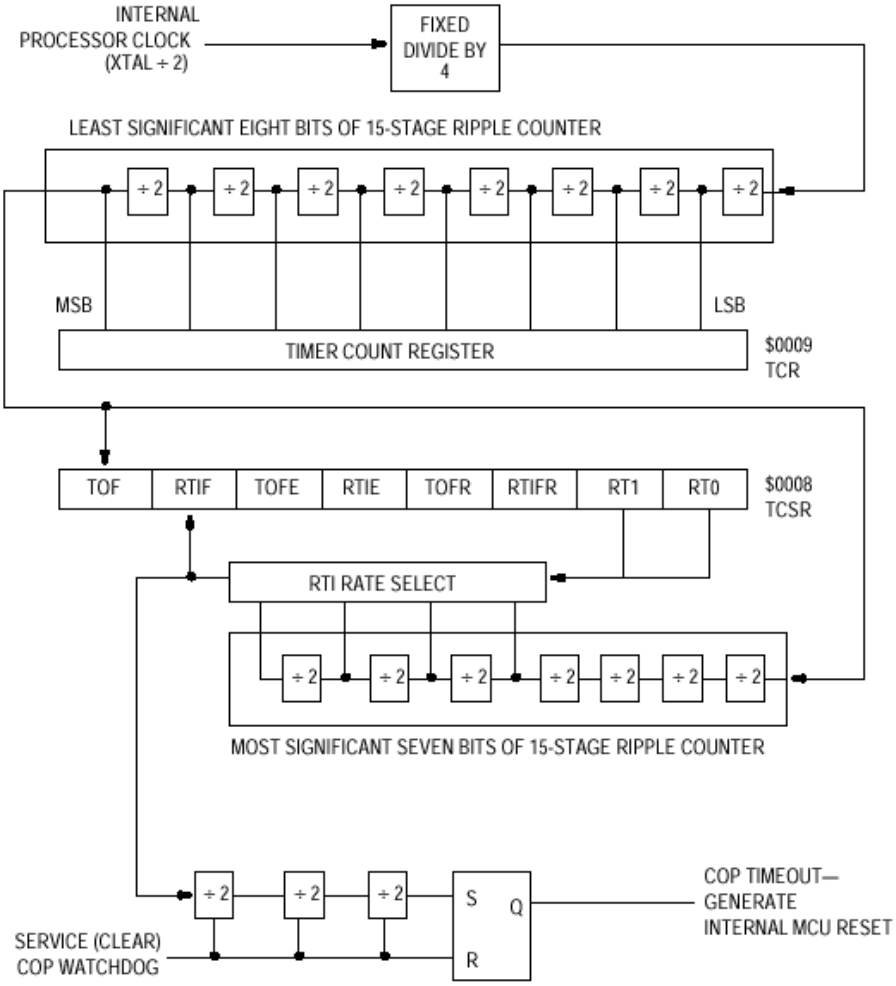


Figura 39. Diagrama de bloques del Temporizador Multifunción de 15 etapas

Además de proporcionar una interrupción periódica, la salida de 8-bits del contador va a la entrada de un contador de 7-bits adicional. La salida de cualquiera de los últimos cuatro bits de este contador, se puede usar para generar una interrupción periódica adicional. Se puede seleccionar uno de estos cuatro bits usando un selector de 4 posiciones, controlado por dos bits RT1 y RT0, en el control del temporizador y en el registro de estado. La Tabla 17 muestra la selección de las cuatro interrupciones de tiempo real disponibles, cuando el microcontrolador trabaja con el E-clock a una frecuencia de 2.0 MHz.

RT1	RT0	RTI	Período de Reset Mínimo del COP
0	0	8.2 ms	57.3 ms
0	1	16.4 ms	114.7 ms
1	0	32.8 ms	229.4 ms
1	1	65.5 ms	458.8 ms

Tabla 17. Timer RTI y COP (E-clock = 2 MHz)

Como etapa final del sistema temporizador multifunción, tiene un contador de 3-bits que forma el sistema de 'watchdog timer' (COP). El sistema COP sirve para proteger contra los fallos de software. Cuando se habilita, se debe realizar una secuencia de resets del COP, antes de que expire el período de 'timeout' del COP evitando un 'reset' de la MCU. Para prevenir que el COP genere un 'reset' en la MCU, se debe escribir a 0 el bit 0 en la posición de memoria $07F0 (COPR), antes de que haya expirado el periodo de reset del COP. Porque la entrada del 'watchdog timer' (COP) está gobernada por la salida del circuito de interrupción de tiempo real, cambiando el tiempo de RTI afectará al mínimo periodo de reset del COP. La Tabla 17 muestra los cuatro periodos de reset del COP disponibles, para los tiempos correspondientes de RTI.

Ejemplo del Temporizador

En esta sección se desarrolla el software que utiliza la interrupción de tiempo real y la interrupción por desbordamiento del temporizador, para producir una señal de baja frecuencia por modulación de ancho de pulso (**PWM**), a través de un pin de E/S de propósito general. Las señales PWM son útiles para una variedad de funciones de control. Pueden controlar la velocidad de un motor o se pueden convertir fácilmente a un nivel de tensión continua (dc) para controlar un dispositivo de salida analógica o formar parte de un convertidor A/D.

Una señal de PWM, como su nombre indica, tiene una frecuencia fija pero varía los anchos de sus estados, alto y bajo. La Figura 40 muestra tres señales PWM con ciclos de trabajo diferentes. Para cada señal, el periodo T1 es constante pero el tiempo del valor alto varía (el periodo de tiempo mostrado por T2). El ciclo de trabajo normalmente se expresa como un porcentaje (la proporción de T2 a T1) y se denomina también **'duty cycle'**.

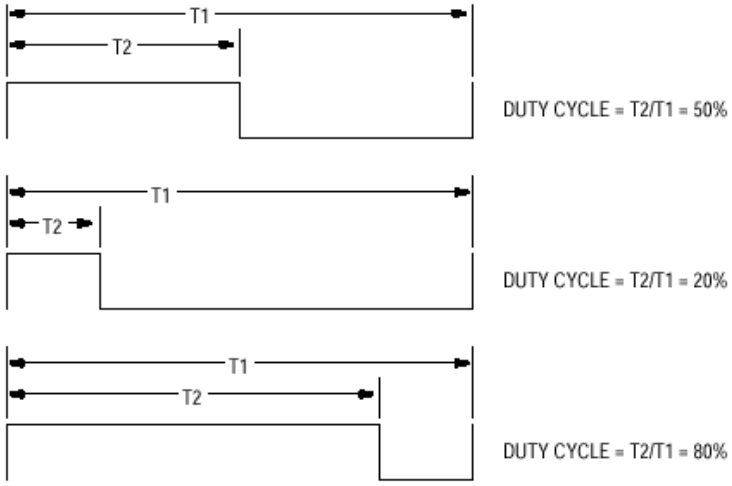


Figura 40. Formas de onda de un PWM con Varios Ciclos de trabajo

Para generar una señal de PWM exacta, se requieren dos referencias de tiempo. Una referencia de tiempo pone la frecuencia constante de la señal de PWM, mientras que la segunda determina la cantidad de tiempo que la salida del PWM permanece en estado 'alto'.

A continuación se describe la estrategia básica de desarrollo de software PWM. Se usará una interrupción de tiempo real (RTIF) para generar el periodo de PWM y se usará el desbordamiento del temporizador (TOF) para determinar el tiempo del estado alto del PWM. El resto de este capítulo es un desarrollo detallado de esta idea básica con una aplicación.

Se empieza mirando el temporizador del MC68HC705J1A. La Figura 41 muestra de nuevo el temporizador para dar énfasis a la porción que se está intentando describir. Conceptualmente, las ocho etapas del contador que se usarán para generar la señal de PWM, rodeadas por un recuadro.

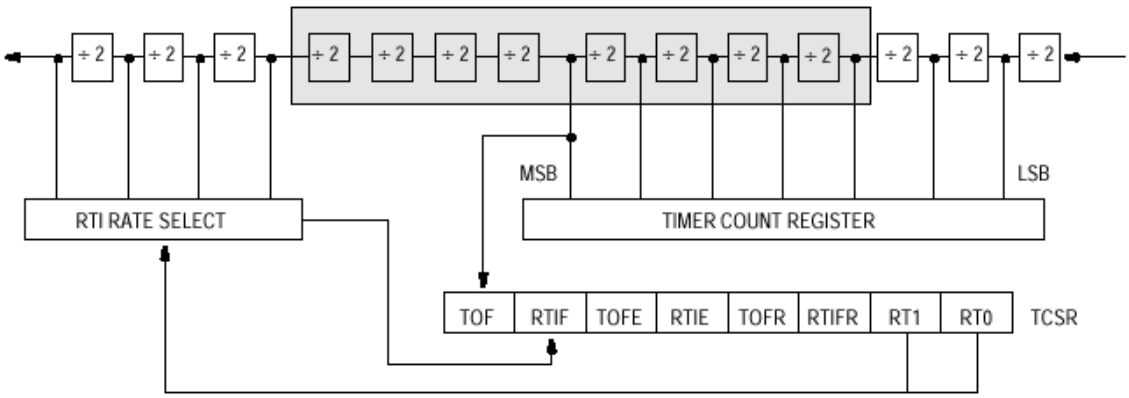


Figura 41. Porción del Temporizador MC68HC705J1A

La Figura 41 muestra cuatro etapas del contador entre la interrupción de salida de desbordamiento del temporizador TOF y la primera entrada del multiplexor de selección de proporción RTI. Esto indica que las

interrupciones de desbordamiento del temporizador ocurrirán a una proporción de 16 veces más rápido que la interrupción de tiempo real seleccionable más rápida.

Usando la RTI para generar la frecuencia base de una señal PWM y la interrupción TOF para determinar el ciclo de trabajo, se podría generar una salida de PWM con 16 ciclos de trabajo discretos (incluyendo 100%) como se muestra en la Figura 42. Los números del lado izquierdo de la figura indican el número de interrupciones TOF que ocurrirán antes de que la salida de PWM se ponga a un nivel bajo. Los números del lado derecho de la figura indican el ciclo de trabajo de la forma de onda. Se puede ver que no hay ninguna interrupción TOF asociada con la forma de onda del 100% de ciclo de trabajo. Como se describirá después, éste es un caso especial que debe probarse en la rutina de RTI.

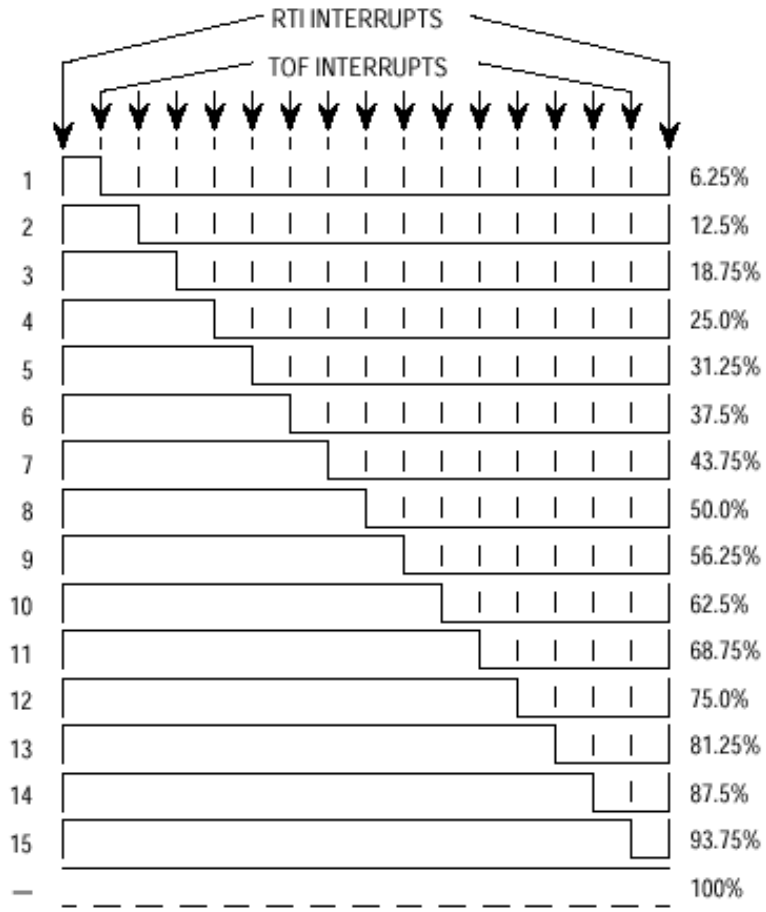


Figura 42. PWM con las 16 salidas de Ciclo de Trabajo

Mientras que el software para llevar a cabo la salida de PWM mostrado es muy simple, teniendo sólo 16 opciones de ancho de pulso, se limita la utilidad de este PWM a un número pequeño de aplicaciones (donde no es necesario un control exacto). Por ejemplo, si se construye un sistema de control de velocidad de un motor usando este PWM, la velocidad sólo se podría controlar al 6.25% (asumiendo que la velocidad del motor es directamente proporcional al promedio del voltaje aplicado). Para la mayoría las aplicaciones de control de velocidad de motor, serían inaceptables con una variación del 12.5% en la velocidad de rotación. Obviamente, es necesario un control mucho más fino del ciclo de trabajo del PWM. Para ello se podría usar una proporción más lenta de la RTI, produciendo un número mayor de interrupciones TOF para cada RTI. En algunas aplicaciones, ésta puede ser una solución aceptable. Sin embargo, para muchas aplicaciones la frecuencia resultante de la forma de onda PWM sería demasiado baja para ser de uso práctico.

La Tabla 18 muestra las proporciones de las cuatro RTI disponibles y la frecuencia de PWM correspondiente, el número de interrupciones TOF entre las RTI y la mínima variación del ciclo de trabajo que sea posible.

Tiempos RTI	Frecuencia PWM	Interrupciones TOF	Mínimo Ciclo de Trabajo
8.2 ms	122 Hz	16	6.25 %
16.4 ms	61.0 Hz	32	3.125 %
32.8 ms	30.5 Hz	64	1.56 %
65.5 ms	15.3 Hz	128	0.78 %

Tabla 18. Características de PWM para diferentes proporciones de la RTI

La Tabla 18 parece sugerir que se está trabajando fuera de frecuencia de PWM, para la exactitud del ciclo de trabajo. Sin embargo, el siguiente programa de software da resultados mucho mejores que los esperados.

Repasando la parte del temporizador de la Figura 41 rodeado por un recuadro, muestra ocho bits de la cadena de 15-bits del temporizador. Cuatro de los bits son accesibles a la CPU, como los cuatro bits superiores del TCR. Los otros cuatro bits forman una cadena del contador que divide por 16, cuyo valor no es directamente accesible. Sin embargo, contando el número de interrupciones TOF que ocurren después de cada RTI, siempre se puede saber el estado de estos cuatro bits del contador. Utilizando un número de 8-bits para representar el ciclo de trabajo del PWM, se puede lograr una exactitud de ciclo de trabajo de $1 \div 255 = 0.4$ %.

Para conseguir este nivel de control con el temporizador del MC68HC705J1A, no se puede usar directamente un valor de ciclo de trabajo de 8 bits. El número de 8 bits debe ser separado en dos componentes. Un componente representa el valor de los cuatro bits inaccesibles por el contador (el número de interrupciones TOF que ocurren después de cada RTI). El otro componente representa el valor de los cuatro bits superiores del TCR (los cuatro bits más bajos del contador que es directamente accesible a la CPU).

Para que el software pueda usar más fácilmente estos dos componentes, los cuatro bits superiores del ciclo de trabajo del PWM deseado, se deben poner en los cuatro bits más bajos de una variable, llamada PWMCoarse (‘coarse’ quiere decir grueso). Este valor será usado durante que interrupción TOF para determinar la salida de PWM que deberá ponerse a un nivel bajo. Los cuatro bits más bajos del ciclo de trabajo del PWM deseado, se pondrá en los cuatro bits superiores de una variable, llamada PWMFine (‘fine’ quiere decir fino). Este valor se usa dentro de la interrupción TOF para precisamente determinar cuando debe ponerse a un nivel bajo la salida del PWM, durante la interrupción TOF. Comparando el valor en PWMFine con los cuatro bits superiores del TCR, se puede dividir eficazmente cada interrupción TOF en 16 intervalos de tiempo separados, como se muestra en la Figura 43.

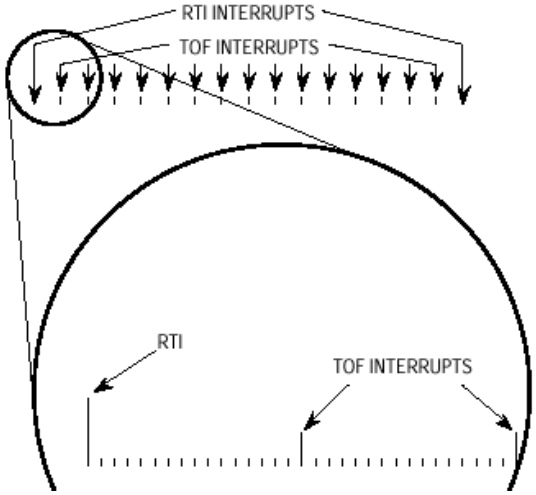


Figura 43. Cada interrupción TOF, troceada en 16 intervalos de tiempo separados.

Ahora que se ha descrito la complicada teoría de la precisa generación de la forma de onda PWM, utilizando el temporizador del MC68HC05J1A, el siguiente paso es escribir el software. Se empieza generando los diagramas de flujo para describir las acciones necesarias para producir la forma de onda PWM y se termina traduciendo los diagrama de flujos en lenguaje ensamblador del MC68HC05.

Los diagrama de flujos de la Figura 44, 45 y 46 describen el Software de PWM. El diagrama de flujo de la Figura 45, aunque es simple, se ha incluido para ver con mayor claridad las acciones de las. Porque el MC68HC05J1A sólo contienen un vector de interrupción del temporizador, una corta rutina debe determinar si una interrupción del temporizador fue causada por una interrupción TOF o RTIF y entonces bifurcar a la rutina de servicio apropiada.

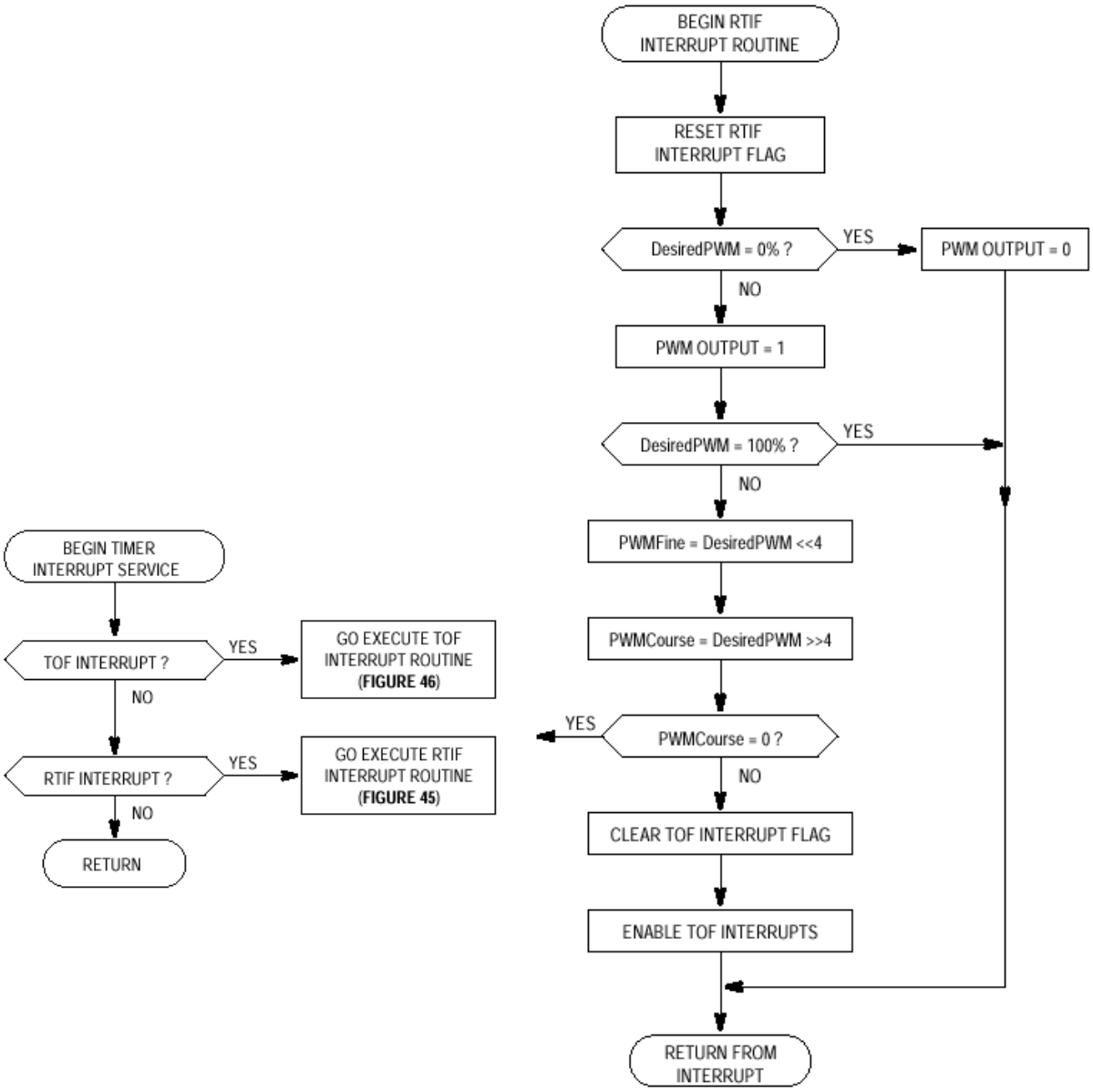


Figura 44. Rutina Servicio de Interrupción del Temporizador

Como se muestra en la Figura 45, las rutinas de interrupción RTIF prueban para dos condiciones especiales, 0% y 100% del ciclo de trabajo. Entonces la introducción de las variables PWMFine y PWMCoarse para ser usadas por la rutina de servicio de interrupción TOF. Si se desea un 0% del ciclo de trabajo, la salida del PWM se pone a un nivel bajo y la rutina de servicio de interrupción RTIF volverá inmediatamente. Si se desea un 100% del ciclo de trabajo, la salida de PWM se pone a un nivel alto y la rutina de servicio de interrupción RTIF volverá inmediatamente. Si se desea un ciclo de trabajo entre el 0% y el 100%, la variable DesiredPWM se reparte en los dos componentes, PWMFine y PWMCoarse. Si el valor resultante de PWMCoarse es 0, el programa saltará a la segunda parte de la rutina de interrupción TOF, qué continuamente compara el valor en PWMFine con los cuatro bits superiores del TCR. Si el valor de PWMCoarse no es 0, se habilitan las interrupciones TOF y vuelve a la rutina de interrupción RTIF.

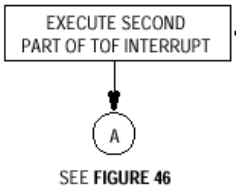


Figura 45. Diagrama de flujo de la Rutina de Interrupción de Tiempo Real

El diagrama de flujo de la Figura 46 describe las acciones requeridas para la rutina de interrupción TOF. La primera acción es decrementar el valor de PWMCoarse. Cuando PWMCoarse vuelve a 0 significa que el valor en los cuatro bits superiores del contador son igual a los cuatro bits superiores de DesiredPWM. A continuación, se comparan continuamente los cuatro bits superiores del TCR con el valor de PWMFine (qué lo forman los cuatro bits más bajos de DesiredPWM). Cuando estos dos valores se igualan, la salida del PWM se pone a un nivel bajo, la interrupción TOF se restablece y se desactiva, y vuelve a la interrupción TOF.

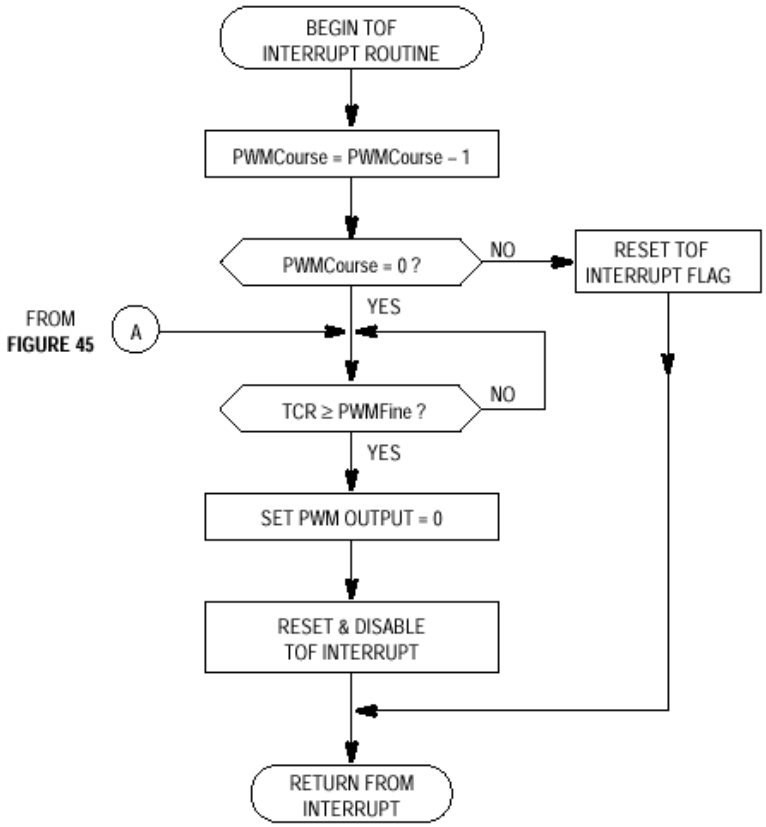


Figura 46. Diagrama de flujo de la interrupción por Desbordamiento del Temporizador

El **Listado 5. Listado del Programa de PWM**, muestras el listado en lenguaje ensamblador de las tres rutinas descritas por los diagramas de flujo de la Figura 44, 45 y 46. La traducción de los diagramas de flujo en lenguaje ensamblador es bastante sencilla. La posible excepción es el código ensamblador en la rutina de interrupción RTIF que se reparte la variable DesiredPWM en las componentes PWMCoarse y PWMFine. Esta rutina trabaja usando una combinación de instrucciones de desplazamiento a la izquierda y rotación a la izquierda, en las que operan los registros A y X. La instrucción LSLA cambia el bit más significativo del registro A en el acarreo y un 0 en el bit menos significativo de A. La instrucción ROLX pone el acarreo (de la anterior instrucción LSLA) en el bit menos significativo del registro X. Después de la ejecución de éstas cuatro pares de instrucciones, los cuatro bits más significativos del registro A (DesiredPWM) terminarán en los cuatro bits menos significativos del registro X (PWMCoarse). Los cuatro bits menos significativos del registro A terminarán en los cuatro bits más significativos del registro A (PWMFine).

Usando el Software PWM

En circunstancias normales, el software PWM del **Listado 5. Listado del Programa PWM**, se usaría como una parte de un programa más grande. El valor de DesiredPWM sería generado por otra parte del programa. Para demostrar el software de PWM, el valor de DesiredPWM se puede poner arbitrariamente a \$80 (128_{10}) por instrucciones del programa. Si se usa un simulador o el emulador para estudiar este programa, se puede cambiar el valor de DesiredPWM y se puede observar el efecto.

El programa de PWM se maneja por interrupciones. Esto significa que el temporizador generará peticiones de interrupción a la CPU, para dejar de procesar el programa principal y responder a la petición de interrupción. Puesto que este programa del **Listado 5. Listado del Programa PWM**, no tiene viene de otro programa principal, se ha incluida una instrucción "branch to here" bifurcación hacia aquí", después de poner a 0 la instrucción de máscara de interrupción (CLI). Esta instrucción es un bucle infinito. Las interrupciones del temporizador causarán a la CPU el dejar periódicamente este bucle infinito, para responder a las peticiones del temporizador y entonces volver a ejecutar el bucle infinito.

Listado 5. Listado del Programa PWM (página 1 de 2)

```
            ;Todas las EQU para el 705J1 están incluidas
            ; pero no se muestran en este listado
            ;
0000        Percent100  EQU   $FF          ;valor de DesiredPWM para 100%
0000        PWM         EQU   PA7          ;salida PWM en bit 7 puerto A
            ;                              ;actualiza la variable
      DesiredPWM.

00C0                    ORG   RAMStart
00C0        DesiredPWM  RMB   1            ;ciclo de servicio PWM deseado...
            ; expresado como el numerador de DesiredPWM/255.
            ; 0 = estado bajo continuo.      255 = estado alto continuo.

00C1        PWMCoarse   RMB   1            ;Numero de interrupciones TOF...
            ; antes se empieza comparando PWMFine con el valor en TCR.

00C2        PWMFine     RMB   1            ;Cuando TCR iguala PWMFine,...
            ;                              ; el PWM se pone 'bajo'.
            ; PWMFine es derivado desde los 4 bits más bajos de DesiredPWM.
            ; Estos 4 bits son puestos en los 4 bits más altos de PWMFine.

00C3        VarEnd      EQU   *

            ;*******************************************************
            ;
0300                    ORG   ROMStart
            ;
0300        Start       EQU   *
0300 9C                 RSP               ;Reset el Stack Pointer
0301 3F00               CLR   PORTA     ;Pone a 0 todas las salidas Puerto
A
0303 A6FF         LDA   #$FF   ;Pone los pins del Puerto A como Salida
0305 B704               STA   DDRA        ;Borra todo de la RAM
0307 AEC0               LDX   #RAMStart   ;Punto del inicio de la RAM
0309 7F     ClrLoop     CLR   ,X          ;Borra un byte
030A 5C                 INCX              ;Apunta a la siguiente posición
                                         ;¿Borrada la última posición?
030B 26FC               BNE   ClrLoop     ;No, Continua borrando la RAM
030D A680               LDA   #$80        ;Corresponde al 50% (128/255)
030F B7C0               STA   DesiredPWM  ;Establece un 'duty cycle' del
PWM
0311 A61C               LDA   #$1C        ;Borra interrupciones del
timer...
0313 B708               STA   TSCR        ;y habilita interrupción RTIF
0315 9A                 CLI               ;Habilita interrupciones
0316 20FE               BRA   *           ;Bucle Infinito, usa PWM
interrup.
```

Listado 5. Listado del Programa PWM (página 2 de 2)

```
            ;*******************************************************
            ;Pone el período RTI. @2MHz y RT1:RT0 = 0:0, período = 8.192 ms
            ;o 122 Hz aproximadamente.
0318        TimerInt    EQU   *
0318 0E0804 BRSET       TOF,TSCR,TOFInt   ;¿interrupción TOF?
```

```
031B 0C0812 BRSET       RTIF,TSCR,RTIInt  ;¿interrupción RTI?
031E 80     RTI

            ;*******************************************************
            ;Respuesta a la interrupción TOF.
            ;Decrementa PWMCoarse, cuando 0...
            ;Compara PWMFine con TCR. Cuando TCR pasa PWMFine borra
            ;el pin de salida PWM y desactiva futuras TOF. Rehabilita RTI.

031F        TOFInt      EQU   *
031F 3AC1               DEC   PWMCoarse   ;¿PWMCoarse = 0?
0321 260A               BNE   ExitTOF     ;No. Borra TOF y vuelve a
0323 B6C2   TOFInt1     LDA   PWMFine     ;comparar los 4 más altos de TCR
0325 B109   CmpMore     CMPA  TCR
0327 22FC               BHI   CmpMore     ;Bucle hasta PWMFine <= TCR
0329 1F00               BCLR  PWM,PORTA   ;Pone la salida PWM a 0 (0V)
032B 1B08               BCLR  TOIE,TSCR   ;Deshabilita la interrupción TOF
032D 1608   ExitTOF     BSET  TOFR,TSCR   ;Reset el Flag de interrupción
TOF
032F 80                 RTI               ;Vuelve al programa principal

            ;*******************************************************
            ;Respuesta a la interrupción RTIF - Pone el pin PWM en alto,
            ;y habilita TOF. Hace PWMCoarse y PWMFine desde DesiredPWM
            ;
0330        RTIInt      EQU   *
0330 1408               BSET  RTIFR,TSCR  ;Borra el Flag de interrupción RT
0332 B6C0               LDA   DesiredPWM  ;¿Conseguido el nivel deseado
                                         ;de PWM. =0?
0334 2719               BEQ   RTIInt2     ;Si. Deja la salida PWM en bajo
0336 1E00               BSET  PWM,PORTA   ;No. Pone la salida PWM en alto
0338 A1FF               CMPA  #Percent100 ;¿Nivel deseado de PWM 100%?
033A 2713               BEQ   RTIInt2     ;Si. Deja la salida PWM en alto
033C 5F                 CLRX              ;No. Pone los 4 bits más altos
033D 48                 LSLA              ;de DesiredPWM en 4 bits bajos
033E 59                 ROLX              ;de A y 4 bits más bajos
033F 48                 LSLA              ;de DesiredPWM en los 4 bits
0340 59                 ROLX              ;más bajos de x.
0341 48                 LSLA
0342 59                 ROLX
0343 48                 LSLA
0344 59                 ROLX
0345 B7C2               STA   PWMFine     ;Guarda resultado en PWMFine.
0347 BFC1   RTIInt1     STX   PWMCoarse   ;Guarda resultado en PWMCoarse.
0349 27D8               BEQ   TOFInt1     ;Si PWMCoarse=0, ir a la segunda
                                         ;mitad de la rutina TOF
034B 1608               BSET  TOFR,TSCR   ;Borra el Falg de desbordamiento
                                         ;del Timer.
034D 1A08               BSET  TOIE,TSCR   ;Rehabilita la interrupción TOF
034F 80     RTIInt2     RTI               ;Vuelve de la interrupción RTIF
07F8                    ORG   Vectors     ;Vectores de interrupción/reset.
07F8 0318               FDB   TimerInt    ;Rutina interrupción del Timer.
07FA 0300               FDB   Start       ;Vector de IRQ (no usado)
07FC 0300               FDB   Start       ;Vector de SWI (no usado)
07FE 0300               FDB   Start       ;Vector de Reset.
```

Ejemplo Práctico de un Control de Motor

En esta sección, se desarrolla una aplicación práctica, expandiendo el software desarrollado en este libro. El ejemplo añade algún hardware externo al MC68HC705K1 para que se puedan observar los efectos del software en el mundo exterior del microcontrolador. También se va a usar una versión ligeramente modificada de la rutina de PWM que se ha desarrollado en este capítulo, para controlar la velocidad de un pequeño motor de corriente continua (dc) de imán permanente. Además, se usan los conceptos desarrollados en el capítulo titulado **Periféricos Internos**, que permiten a la CPU leer el estado de los interruptores conectados a los pins de E/S de la MCU de propósito general.

Teoría

Los motores de continua (DC) son a menudo la mejor opción para aplicaciones de variación de velocidad de motores. Los motores de continua con escobillas son los más fáciles de controlar electrónicamente. El control electrónico de motores sin escobillas (Brushless), paso a paso (Stepper), inducción (CA) y reluctáncia variable (SR), requieren circuitos de control más complejos, además de los dispositivos conmutación de potencia. Los pequeños motores con escobillas (DC) de bajo costo están disponibles en muchas aplicaciones, donde los diseños hechos a medida serían demasiado caros. La fiabilidad de los motores con escobillas es adecuada para la mayoría de las aplicaciones. Sin embargo, en el futuro, las escobillas se gastan y necesitan ser reemplazadas.

Para variar la velocidad de un motor de continua con escobillas, se debe variar el voltaje que se aplica al motor. Se pueden usar varias maneras para lograr esto. Se examinarán algunos métodos y se explicarán la mayoría de ventajas y desventajas de cada uno.

El primer método más obvio para variar el voltaje aplicado a un motor, podría ser poner una resistencia variable en serie con el motor y la fuente alimentación, como se muestra en la Figura 47. Este método es muy simple, pero tiene algunas serias desventajas. Primero, la disipación de potencia en la resistencia se debe acercar a los requerimientos de la potencia del motor. Para pequeños motores que no llegan al caballo de potencia, el tamaño de la resistencia variable sería bastante modesto. Sin embargo, con el aumento del tamaño del motor, el requerimiento de potencia del motor aumenta y también el tamaño y el costo de la resistencia variable.

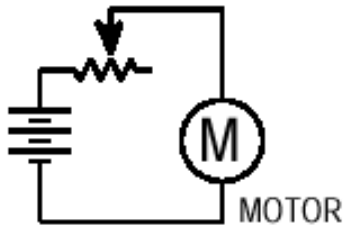


Figura 47. Control de Velocidad a través de una Resistencia Variable

La segunda desventaja de este tipo de control de velocidad es la incapacidad para ajustar la velocidad del motor automáticamente, para compensar las variaciones de carga. Ésta es la primera desventaja para las aplicaciones que requieren un control de velocidad preciso bajo variaciones de carga mecánicas.

En la Figura 48 se muestra un variador electrónico para el control de velocidad de un motor, de la misma forma que con una resistencia variable. En esta figura, se ha reemplazado la resistencia variable con un transistor. Aquí, el transistor se trabaja en modo lineal. Cuando un transistor trabaja en este modo, se comporta esencialmente como una resistencia variable eléctricamente controlada. Aplicando una señal proporcional, del control analógico, al transistor, la "resistividad" del transistor se puede variar y a su vez se variará la velocidad del motor. Usando de esta manera un transistor para controlar la velocidad del motor, la magnitud de la señal de control se reduce a un voltaje mucho más bajo y a niveles de corriente que pueden ser generados fácilmente por una circuitería electrónica.

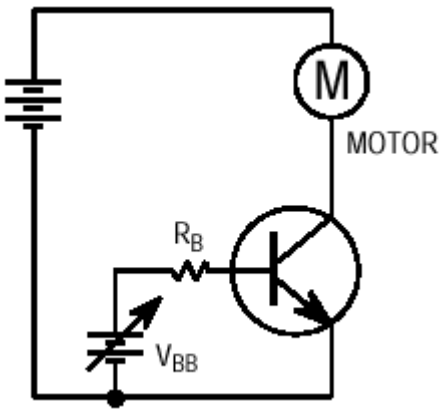


Figura 48. Velocidad controlada por un Transistor

Desgraciadamente, todavía usando un transistor en su modo lineal tiene una desventaja mayor que usando una resistencia variable. Igual que con una resistencia variable, un transistor de potencia que trabaja en su región lineal, tendrá que disipar gran cantidad de potencia bajo variaciones de velocidad y condiciones de carga. Incluso aunque los transistores de potencia son capaces de manejar altos niveles de potencia y que están disponibles a precios relativamente modestos, la potencia disipada por el transistor normalmente exigirá un disipador de calor grande, para prevenir que se destruya el dispositivo.

Además de trabajar como un dispositivo lineal, los transistores pueden estar también trabajando como interruptores electrónicos. Aplicando la señal de control apropiada a un transistor, el dispositivo conducirá o no conducirá. Como se muestra en la Figura 49, cuando el transistor conduce se comportará esencialmente como un interruptor mecánico, que permite que la corriente eléctrica atraviese libremente la carga. Cuando no conduce, no pasa ninguna corriente a través del transistor o carga. Porque el transistor disipa muy poca potencia cuando conduce totalmente o se satura, el dispositivo trabaja de una manera eficaz.

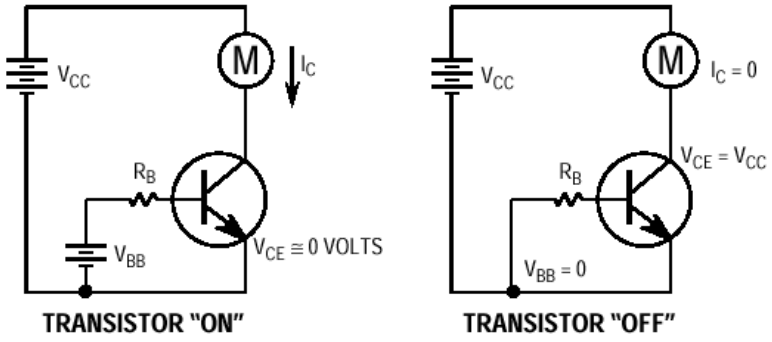


Figura 49. Interruptor Electrónico usando un Transistor

Cuando se usa un transistor para controlar la velocidad de un motor de continua, usando un dispositivo en su modo ineficazmente lineal si se quiere que un motor trabaje plena velocidad. Afortunadamente, hay un método alternativo para controlar la velocidad de un motor de continua usando un transistor. Usando el transistor como un interruptor controlado electrónicamente y aplicando una señal de control PWM de suficiente frecuencia, para controlar la velocidad del motor. Para ayudar a entender cómo se puede controlar la velocidad de un motor a su total velocidad y a ninguna, hay que considerar las formas de onda de PWM de la Figura 50.

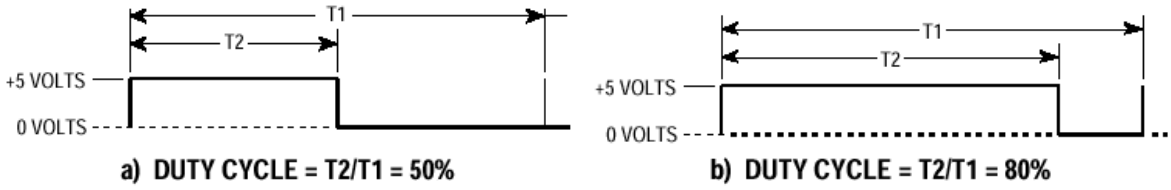


Figura 50. Formas de Onda de PWM con el 50 % y 80 % del Ciclo de Trabajo

La Figura 50(a) muestra un único ciclo, de una forma de onda del 50 % del ciclo de trabajo, que es de 5 V durante la primera mitad de su periodo y de 0 V durante el segundo medio periodo. Si se integra (o se promedia) el voltaje de la forma de onda de PWM en la Figura 50(a) sobre este periodo, T1, el promedio de voltaje continuo es del 50 % de 5 V, que resulta 2.5 V. Correspondientemente, el promedio de voltaje continuo de la forma de onda de PWM de la Figura 50(b) que tiene un ciclo de trabajo del 80 %, es 80 % de 5 V, que resulta 4.5 V. Usando una señal de PWM para conmutar un motor en marcha y paro, producirá el mismo efecto que aplicar al motor un voltaje continuo o promedio de los diferentes niveles. La frecuencia de señal PWM debe ser suficientemente alta para que la inercia rotatoria del motor integrada del pulso on/off, provoque que el motor ruede fácilmente.

Circuito de Control de un motor

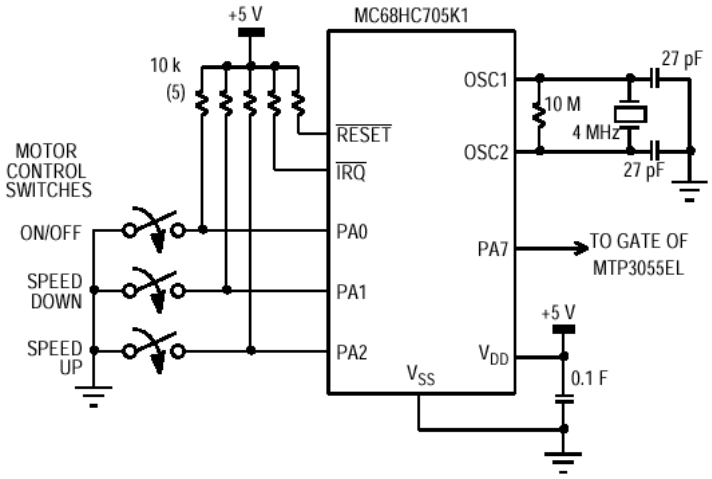
Como se mencionó antes, se está usando una versión ligeramente modificada de la rutina de PWM para controlar la velocidad de un pequeño motor. Sin embargo, antes de discutir el software involucrado, se necesita echar una mirada a los componentes hardware requeridos para manejar el motor.

La Figura 52 es un esquema de la etapa de potencia del circuito control de motor. Hay varias diferencias entre este esquemático y los que se usaron en la Figura 48 y 49. Se describirán estas diferencias en los párrafos siguientes.

La diferencia más notable del esquemático es el transistor de potencia que se usará como interruptor electrónico. Este dispositivo es un MOSFET de potencia. Al contrario del transistor bipolar mostrado en la Figura 48 y 49, este tipo especial de transistor es controlado por el voltaje aplicado a su puerta. Adicionalmente, este MOSFET de potencia, el MTP3055EL, se puede saturar completamente con sólo 5 V aplicados a su puerta. Estas dos características permiten a este dispositivo ser controlado directamente por el pin de salida de un microcontrolador para muchas aplicaciones.

Ya que la impedancia de entrada de un MOSFET de potencia es muy alta (mayor que 40 megaohms), se pone una resistencia de 10KΩ entre la puerta del MOSFET y tierra, para evitar el funcionamiento errático del motor la conexión entre el microcontrolador y la puerta ser cortada. El diodo zener de 15 V se pone en paralelo con la resistencia, para proteger la puerta del MOSFET de algún posible daño debido a algún transitorio de alto voltaje que pueden generarse en el sistema. El diodo 1N4001 en paralelo con el motor se usa para amortiguar las puntas inductivas del motor cada vez que el MOSFET deja de conducir. El condensador de 0,1 µF en paralelo con el motor se usa para reducir el ruido eléctrico generado por las escobillas del motor.

La Figura 51 muestra un esquema del circuito microprocesador que se usará en este ejemplo. Además de generar una salida de PWM, el MC68HC705K1 lee tres pulsadores conectados a sus pins de E/S. Como muestra el esquema, un simple pulsador pone en marcha y para el motor, mientras que los otros dos pulsadores ponen la velocidad del motor.



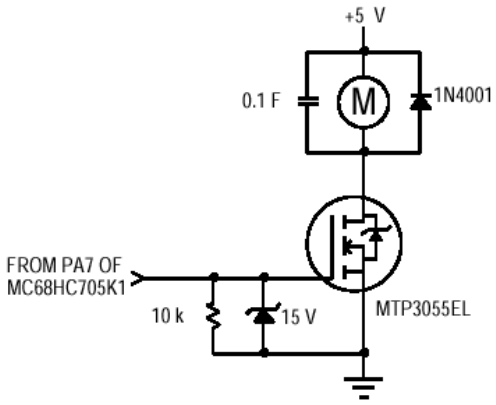


Figura 51. Microcontrolador Controlador de Velocidad del Motor.

Figura 52. Etapa de Potencia

Un lado de cada pulsador está conectado a tierra, mientras que el otro lado está conectado a un pin de E/S del microcontrolador MC68HC705K1. Cada uno de los pins de entrada del microcontrolador tienen conectada una resistencia de 10 kΩ a +5 V. Estas resistencias de 10 kΩ (llamadas de 'pullup'), ponen los tres pins de entrada a un nivel lógico 1 cuando los pulsadores no están pulsados.

En este circuito de ejemplo, el control de capa pulsador opera de la siguiente manera. El pulsador de marcha paro ('on/off') del motor, trabaja como un control de acción alternativa. Cada vez que el pulsador se pulsa y se suelta, el motor alternativamente se pone en marcha o se para. Cuando el motor se pone en marcha, su velocidad se pondrá a la velocidad que iba la última vez en la que el motor estuvo encendido.

Los pulsadores de la velocidad 'Speed Up' y 'Speed Down' hacen aumentar o disminuir respectivamente la velocidad del motor. Para aumentar o disminuir la velocidad del motor, el respectivo pulsador se debe apretar y se debe mantener. La velocidad del motor con modulación de ancho de pulsos (PWM) aumentará o disminuirá a una proporción aproximada de 0.4 % cada 24 ms. Esta proporción o "rampa", permitirá ajustar la velocidad del motor a través de su rango de velocidad completa, en aproximadamente seis segundos.

Software del Control de Motor

La Figura 53 muestra un diagrama de flujo que describe la nueva interrupción RTI software. El único cambio funcional de la rutina de PWM desarrollada antes en este capítulo, es añadir una instrucción al principio de la rutina de servicio de interrupción RTI. Esta instrucción decrementa la variable RTIDlyCnt. Esta variable se usa para las tres rutinas que leen la entrada de los pulsadores para desarrollar un retardo contra los rebotes de los pulsadores.

Como se mencionó en el capítulo de **Programación**, normalmente hay muchas maneras de realizar una tarea específica usando el juego de instrucciones del microcontrolador. Para demostrar esto, una parte de la rutina de interrupción RTI revisada, tiene implementada una ligera diferencia. Se puede revisar el **Listado 6. Listado del Programa de Control de Velocidad**, que se tenía compartida la variable DesiredPWM en dos partes, la variable PWMFine y la variable PWMCoarse. Para hacer esto, se usa una combinación de instrucciones de desplazamientos y de rotaciones para poner los cuatro bits más altos del acumulador A (DesiredPWM) en los cuatro bits más bajos del registro X (PWMCoarse) y los cuatro bits más bajos del acumulador A en los cuatro bits más altos del acumulador A (PWMFine). Este método requiere nueve bytes de memoria de programa y 26 ciclos de CPU.

Usando la alternativa del **Listado 6. Listado del Programa de Control de Velocidad**, se puede conseguir el mismo resultado en sólo tres bytes de memoria de programa y 13 ciclos de CPU.

La rutina RTIInt en el **Listado 6. Listado del Programa de Control de Velocidad** demuestra la alternativa. La secuencia original de la instrucción de 9-bytes, se ha reemplazado con dos instrucciones, LDX #16 y MUL. La instrucción MUL multiplica el valor del acumulador por el valor en el registro de índice y pone el resultado en X:A (encadenamiento de X y A). Multiplicando un número binario por 16 es equivalente a desplazar el valor izquierdo por cuatro posiciones. Igual como en la aplicación original, los cuatro bits más altos de DesiredPWM están ahora en los cuatro bits más bajos del registro X (PWMCoarse) y los cuatro bits más bajos del registro A se han movido en los cuatro bits más altos del registro A (PWMFine).

El diagrama de flujo de la Figura 54 describe la rutina del bucle principal del módulo de control de motor. Este módulo verifica el estado de cada una de las tres entradas de los pulsadores. Si se pulsa cualquiera de los tres pulsadores, se llama a una rutina que maneja estas acciones para este pulsador. Si no hay ningún pulsador pulsado, el bucle principal se repite.

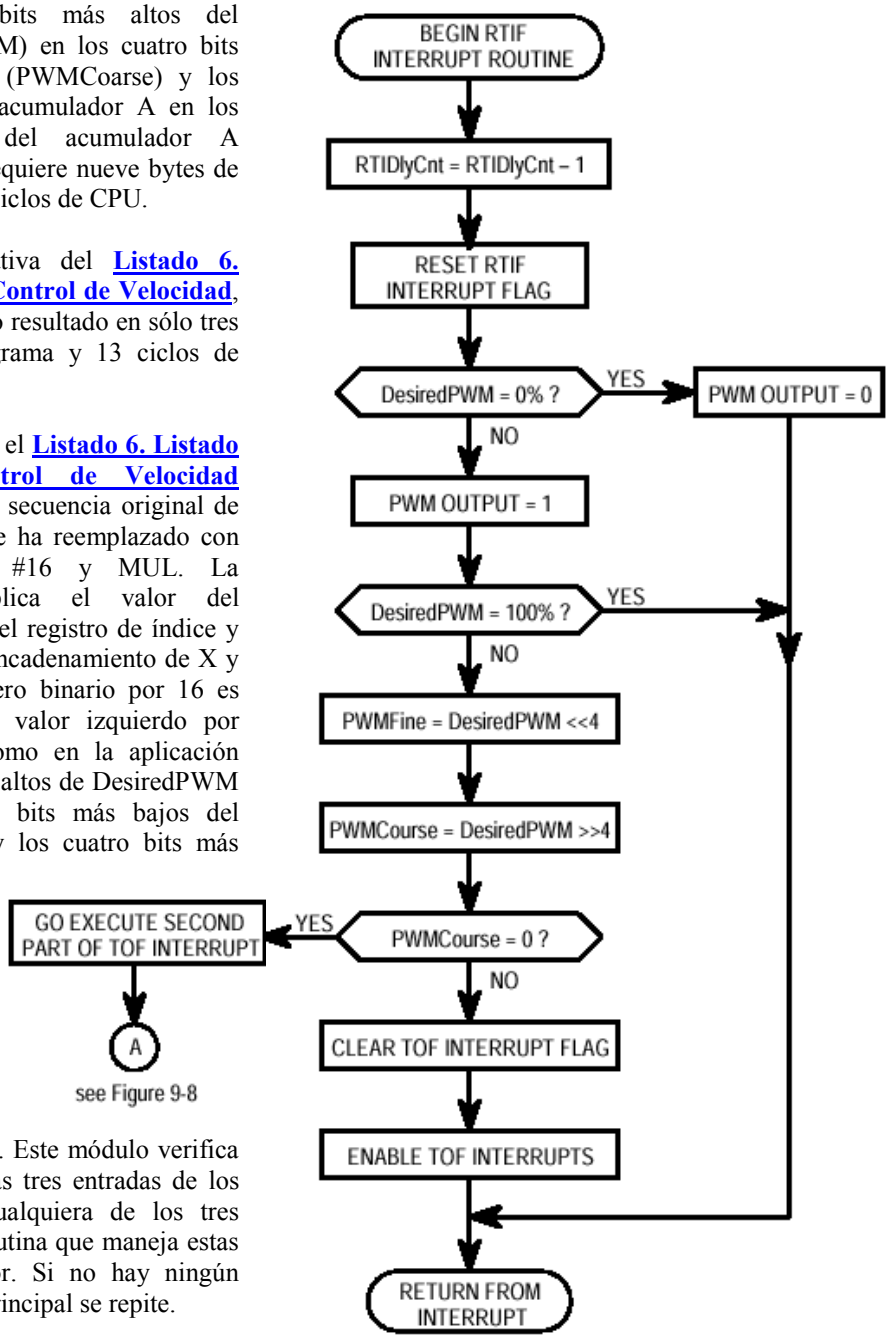


Figura 53. Diagrama de Flujo de la Rutina RTI Revisada

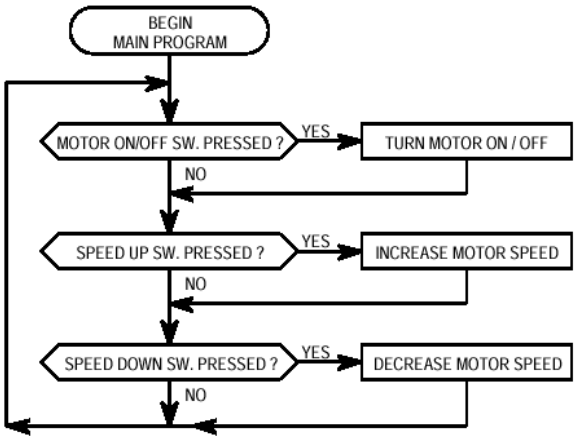


Figura 54. Diagrama de Flujo para el Bucle del Programa Principal

Las Figuras 55, 56 y 57 son diagramas de flujo para las tres rutinas que manejan las acciones de los tres pulsadores de entrada. Cada una de estas rutinas empiezan con la ejecución de una rutina contra rebotes del pulsador de 50 ms. Como se ha descrito en el capítulo de **Programación**, este retardo se necesita porque el salto mecánico producido por el cierre de un pulsador es visto por el microcontrolador como múltiples cierres del pulsador, durante los primeros milisegundos después de apretar el pulsador. Esta pequeña sección de código guarda el valor DebounceDly en la variable RTIDlyCnt y entonces el valor de espera es decrementado hasta ponerse a cero por la rutina de servicio de interrupción. Cuando alcanza el valor cero, el pulsador se verifica otra vez para asegurar de nuevo el cierre del pulsador. El valor usado para el retardo constante (DebounceT), producirá un retardo mínimo, aproximadamente de 50 milisegundos.

El diagrama de flujo de la Figura 55 describe la rutina de Motor On/Off. Es responsable de manejar las acciones de la acción alternativa del pulsador que pone el motor en marcha y paro. Después del retardo contra los rebotes del pulsador, esta rutina espera hasta que el pulsador on/off se deje de pulsar, antes de que realice el resto de su tarea y vuelva al bucle principal. Por otra parte, el bucle principal detectará otro cierre del pulsador en cuanto el programa de MotorOnOff termine y vuelva al bucle del programa principal.

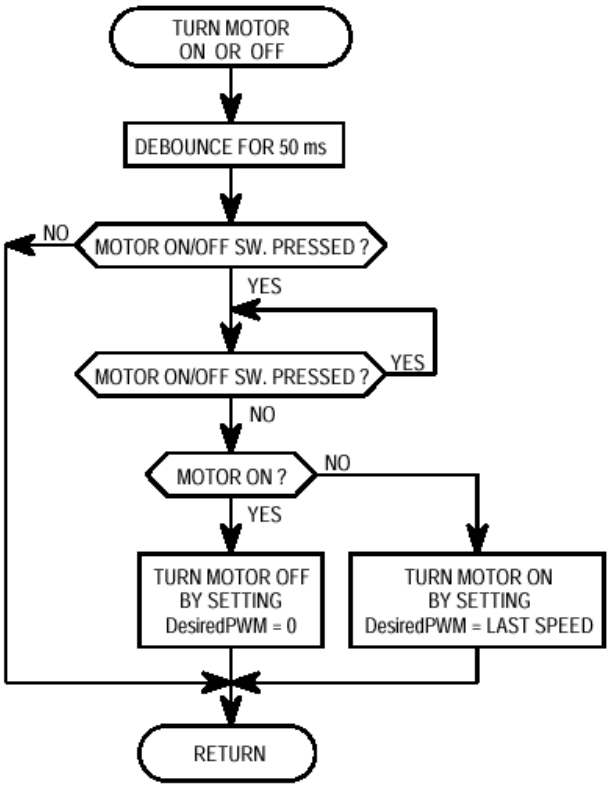


Figura 55. Organigrama de la Rutina de control de Motor 'on/off'

Las rutinas descritas por los diagramas de flujo de la Figura 56 y 57 trabajan esencialmente de la misma manera. Primero, cada uno de estas rutinas prueba si el motor está en marcha. Si el motor está parado, la rutina vuelve al bucle del programa principal. Entonces cada rutina hace el bucle continuamente tantas veces como es asociado al pulsador que continua pulsado. Cada vez a través del bucle, las variables MotorPWM y DesiredPWM son incrementadas o decrementadas para aumentar o disminuir el ciclo de trabajo de salida PWM. Para impedir que la velocidad del motor aumente o disminuya demasiado rápida, cuando se aprieta un pulsador, se inserta un retardo de 25 ms aproximadamente cada vez a través del bucle. Este retraso de 25 ms permite ajustar el motor por su rango completo de velocidad, en 6 segundos aproximadamente.

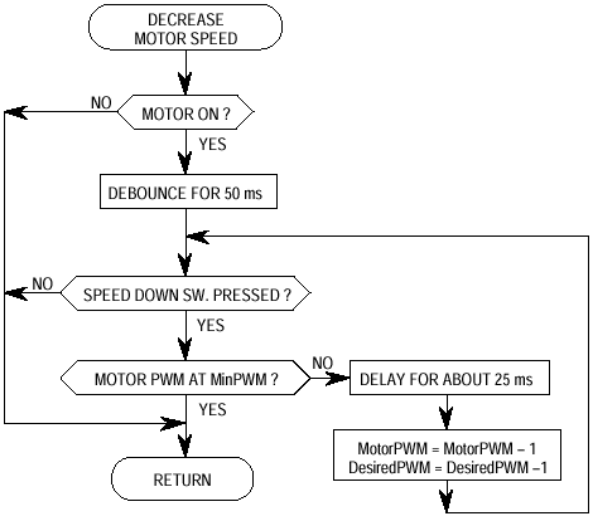


Figura 56. Rutina de Motor 'Speed Up'

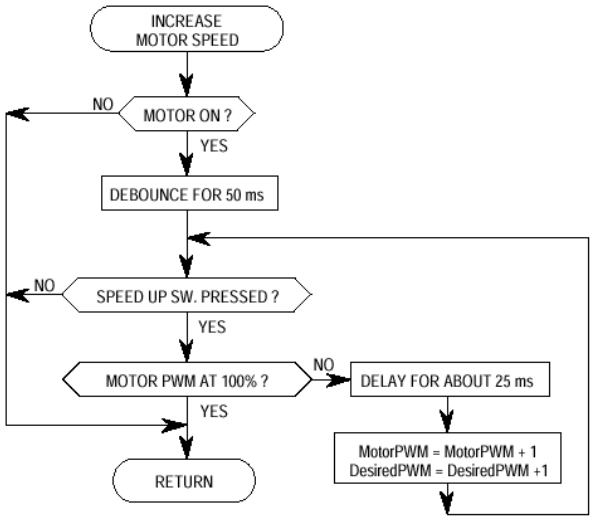


Figura 57. Rutina de Motor 'SpeedDown'

El **Listado 6. Listado del Programa de Control de Velocidad**, contiene el listado del lenguaje ensamblador para las rutinas descritas para los diagramas de flujo de la Figura 46 y 53 a través de la Figura 57.

Listado 6. Listado del Programa de Control de Velocidad (página 1 de 4)

```
            ;Todas las EQU para 705K1 están incluidas, pero no se muestran
00FF        Percent100  EQU   $FF          ;Valor de DesiredPWM para
100%duty
0003        RampTime    EQU   3            ;Rampa constante Speed Up/Down
0007        DebounceT   EQU   7            ;Rebote del Pulsador constante
0010        MinPWM      EQU   $10          ;Mínimo valor PWM.
0007        PWM         EQU   PA7          ;Bit 7 Puerto A es salida PWM
0000        MotorOnOff  EQU   PA0          ;Pulsador para Motor On/Off
0001        SpeedUp     EQU   PA1        ;Pulsador para aumentar la
velocidad
0002        SpeedDn     EQU   PA2          ;Pulsador para bajar la velocidad
00E0                    ORG   RAMStart
00E0        DesiredPWM  RMB   1            ;Deseado PWM/255 = duty cycle
                                          ;0 = bajo continuo
                                          ;255 = alto continuo
00E1        PWMCoarse   RMB   1            ;Numero de TOFs antes...
                                          ;inicia observando PWMFine vs TCR
00E2        PWMFine     RMB   1            ;Cuando TCR iguala a PWMFine,...
                                          ;pone la salida PWM a bajo
00E3        MotorPWM    RMB   1            ;Ultima PWM/vel. mientras
motor.on
00E4        RTIDlyCnt   RMB   1            ;Decrementando en cada RTI...
                                          ;usado para rebote de pulsador
00E5        MotorOnFlg  RMB   1            ;1 = salida PWM On / 0 = off
00E6        VarEnd      EQU   *

            ;*******************************************************
0200                    ORG   ROMStart
0200        Start       EQU   *
0200 9C                 RSP               ;Reset el Stack Pointer en
caso...
                                          ;de recibir aquí un error
0201 3F00               CLR   PortA       ;Pone a 0 todo el Puerto A
0203 A680               LDA   #$80        ;Hace PA7 salida
0205 B704               STA   DDRA        ;Borra toda salida de RAM
0207 AEE0               LDX   #RAMStart   ;Punto de inicio de la RAM
0209 7F     ClrLoop     CLR   0,x         ;Borra un byte.
```

Listado 6. Listado del Programa de Control de Velocidad (página 2 de 4)

```
020A 5C                 INCX              ;¿Apunta a siguiente pos./ hecho?
020B 26FC               BNE   ClrLoop     ;No; continua borrar la RAM
020D A61C               LDA   #$1C        ;Habilita interrup. TOF y RTI
020F B708               STA   TSCR
0211 A610               LDA   #MinPWM     ;Inicializa PWM a la mín. veloc.
0213 B7E3               STA   MotorPWM
0215 9A                 CLI               ;Habilita interrupciones

            ;*********************************************************
            ;Bucle del programa principal.
            ;Lee los pulsadores del control motor.
            ;Si un pulsador es pulsado, BSR realiza la acción pedida.
            ;El Bucle continuamente vigila el cierre de los pulsadores.

0216 0000 02 Main       BRSET MotorOnOff,PortA,Main1 ;¿Pulsado On/Off?
0219 AD0C               BSR   DoOnOff     ;Si es SI, ir a DoOnOff
021B 0200 02 Main1      BRSET SpeedUp,PortA,Main2 ;Speed Up Pulsado?
021E AD25               BSR   DoSpeedUp   ;Si es SI, ir a DoSpeedUp
0220 0400 F3 Main2      BRSET SpeedDn,PortA,Main ;¿Baja la velocidad?
0223 AD44               BSR   DoSpeedDn   ;Si es SI, ir a DoSpeedDown
0225 20EF               BRA   Main        ;Repetir el bucle continuamente

            ;*********************************************************
            ;DoOnOff maneja los rebotes del cierre del pulsador On/Off
            ;del Motor y espera que se deje de pulsar.

0227        DoOnOff     EQU   *
0227 A607               LDA   #DebounceT  ;DebounceT*RTI  = 50mS
0229 B7E4               STA   RTIDlyCnt   ;Inicializa contador software
022B 3DE4   DoOnOff1    TST   RTIDlyCnt   ;interrupción RTI decrementa este
022D 26FC               BNE   DoOnOff1    ;Bucle hasta RTIDlyCnt = 0
022F 0000 12            BRSET MotorOnOff,PortA,DoOnOff3 ;Entonces prueba
                                          ;si el pulsador está abierto,
                                          ;no ha habido buena pulsación.
0232 0100 FD            BRCLR MotorOnOff,PortA,* ;Espera para pulsador está
                                                 ;dejado de pulsar
0235 3DE5               TST   MotorOnFlg  ;¿Motor aún en marcha?
0237 2607               BNE   DoOnOff2    ;Si, para el motor.
0239 3CE5               INC   MotorOnFlg  ;No, pone el flag 'MotorOn'
023B B6E3               LDA   MotorPWM    ;y coge la última veloc. del
motor
023D B7E0               STA   DesiredPWM  ;Activa la salida PWM
023F 81                 RTS               ;Vuelve (1 de 2)
0240 3FE0   DoOnOff2    CLR   DesiredPWM  ;Desactiva la salida PWM
0242 3FE5               CLR   MotorOnFlg  ;Borra el flag 'MotorOn'
0244 81     DoOnOff3    RTS               ;Vuelve (2 de 2)

            ;*********************************************************
            ;DoSpeedUp maneja los rebotes del cierre del pulsador Speed Up
            ; incrementa el ciclo de trabajo hasta que se deje de pulsar
            ; El ciclo de servicio se incrementa aprox. 24 ms.
            ; Ajusta a través de todo el rango de velocidad aprox 6 seg.

0245        DoSpeedUp   EQU   *
0245 3DE5               TST   MotorOnFlg  ;¿Motor en marcha?
0247 2601               BNE   DoSpeedUp2  ;Si, bifurca
0249 81     DoSpeedUp1  RTS               ;No, los pulsadores no trabajan
024A A607   DoSpeedUp2  LDA   #DebounceT  ;Retardo de rebote (aprox 50 ms)
024C B7E4               STA   RTIDlyCnt   ;Inicializa el contador software
024E 3DE4   DoSpeedUp3  TST   RTIDlyCnt   ;interrupción RTI decrementa éste
0250 26FC               BNE   DoSpeedUp3  ;bucle hasta RTIDlyCnt = 0
0252 0200 F4 DoSpeedUp4 BRSET SpeedUp,PortA,DoSpeedUp1 ;RTS si Pulsador Off
0255 B6E3               LDA   MotorPWM    ;Pulsador pulsado, subir veloc.
0257 A1FF               CMPA  #Percent100 ;¿ha total velocidad?
0259 27EE               BEQ   DoSpeedUp1  ;Si es SI, volver
025B A603               LDA   #RampTime   ;No, coge rampa retardo de tiempo
                                          ;(3 * 8.2Ms = 24.6)
025D B7E4               STA   RTIDlyCnt   ;Guarda contador software
025F 3DE4 DoSpeedUp5    TST   RTIDlyCnt   ;¿Rampa retardo tiempo expirado?
```

Listado 6. Listado del Programa de Control de Velocidad (página 3 de 4)

```
0261 26FC               BNE   DoSpeedUp5  ;No, continua esperando
0263 3CE3               INC   MotorPWM    ;Si, incrementa velocidad motor
0265 3CE0               INC   DesiredPWM  ;Adelanta el valor deseado de PWM
0267 20E9               BRA   DoSpeedUp4  ;Bucle para pulsador,
                                          ;hasta pulsado

            ;*********************************************************
            ;DoSpeedDn maneja el cierre del pulsador Speed Down
            ;Rebotes de pulsador incrementan el duty cycle hasta soltar.
            ; Duty cycle incrementado aprox cada 24 ms.
            ; Ajusta a través del rango completo de veloc. en aprox 6 seg

0269        DoSpeedDn   EQU   *
0269 3DE5               TST   MotorOnFlg  ;¿Motor en marcha?
026B 2601               BNE   DoSpeedDn2  ;SI, bifurca
026D 81     DoSpeedDn1  RTS               ;No, pulsador no está pulsado
026E A607   DoSpeedDn2  LDA   #DebounceT  ;Rebotes retardan aprox 50 ms
0270 B7E4               STA   RTIDlyCnt   ;Inicializa software del contador
0272 3DE4   DoSpeedDn3  TST   RTIDlyCnt   ;interrupción RTI decrementa éste
0274 26FC               BNE   DoSpeedDn3  ;Lazo hasta RTIDlyCnt = 0
0276 0200 F4 DoSpeedDn4 BRSET SpeedUp,PortA,DoSpeedDn1
                                          ;RTS si no es pulsado
0279 B6E3               LDA   MotorPWM    ;Pulsado, acelera.
027B A110               CMPA  #MinPWM     ;¿aún a mínima velocidad?
027D 27EE               BEQ   DoSpeedDn1  ;Si es SI, volver.
027F A603               LDA   #RampTime   ;No, rampa retardo(3*8.2ms =24.6)
0281 B7E4               STA   RTIDlyCnt   ;Guarda contador software.
0283 3DE4   DoSpeedDn5  TST   RTIDlyCnt   ;¿Rampa de retardo expirada?
0285 26FC               BNE   DoSpeedDn5  ;No, continua la espera.
0287 3AE3               DEC   MotorPWM    ;Si, decrece la veloc. del motor
0289 3AE0               DEC   DesiredPWM  ;Reduce valor deseado PWM.
028B 20E9               BRA   DoSpeedDn4  ;Lazo para pulsador aún pulsado.

            ;*********************************************************
            ;En las interrupciones RTI y TOF participa 1 vector, TimerInt
            ;se usa para decidir que fuente de servicio fue pedida.
            ;Las rutinas de servicio TOFInt y RTIInt, ambas son usadas
            ;para generar una señal PWM.

028D        TimerInt    EQU   *
028D 0E08 04            BRSET TOF,TSCR,TOFInt   ;¿interrupción TOF?
0290 0C08 12            BRSET RTIF,TSCR,RTIInt  ;¿interrupción RTI?
0293 80                 RTI                     ;No debe tener aquí
                                                ;(código defensivo)

            ;*********************************************************
            ;Respuesta a interrupción TOF - Decrementa PWMCoarse, cuando
            ;0... compara PWMFine a TCR. Cuando pasa TCR, PWMFine borra
            ;el pin de salida PWM y desactiva futuras TOF. RTI re-activa.
            ;
0294        TOFInt      EQU   *
0294 3AE1               DEC   PWMCoarse   ;¿PWMCoarse = 0?
0296 260A               BNE   ExitTOF     ;No. Borra TOF y vuelve
0298 B6E2   TOFInt1     LDA   PWMFine     ;compara los 4 superiores de TCR
029A B109   CmpMore     CMPA  TCR


029C 22FC               BHI   CmpMore     ;Lazo hasta PWMFine <= TCR
029E 1F00               BCLR  PWM,PortA   ;Pone la salida PWM = 0V
02A0 1B08               BCLR  TOIE,TSCR   ;Desactiva la interrupción TOF
02A2 16 08  ExitTOF     BSET  TOFR,TSCR   ;Reset el Flasg de interr. TOF
02A4 80                 RTI               ;Vuelve al programa principal

            ;*********************************************************
            ;Respuesta a interrupción RTIF - Pone el pin PWM = 1, y
            ;activa TOF. Hace PWMCoarse y PWMFine desde DesiredPWM
            ;
02A5        RTIInt      EQU   *
02A5 3AE4               DEC   RTIDlyCnt   ;RTIDlyCnt = RTIDlyCnt - 1.
```

Listado 6. Listado del Programa de Control de Velocidad (página 4 de 4)

```
02A7 1408               BSET  RTIFR,TSCR  ;Borra el Flag interr. RT
02A9 B6E0               LDA   DesiredPWM  ;¿Logra el nivel deseado PWM = 0?
02AB 2603               BNE   RTIInt2     ;No,. Pone la salida = 1
02AD 1F00               BCLR  PWM,PortA   ;Pone la salida = 0, duty es 0%
02AF 80                 RTI               ;Vuelve de la interrupción
02B0 1E00 RTIInt2       BSET  PWM,PortA   ;Salida PWM =1, duty > 0%
02B2 A1FF               CMPA  #Percent100 ;¿És PWM deseado duty = 100%?
02B4 270D               BEQ   RTIInt3     ;Si, Salida siempre = 1
02B6 AE10               LDX   #16         ;No, Pone 4-bits superiores de
02B8 42                 MUL               ;DesiredPWM en los 4-bits bajos
                                          ;de X y los 4bits bajos de
                                          ;DesiredPWM en los 4-bits
                                          ;superiores de A.
02B9 B7E2               STA   PWMFine     ;Guarda resultado PWMFine
02BB BFE1               STX   PWMCoarse   ;Guarda resultado en PWMCoarse
02BD 27D9               BEQ   TOFInt1     ;Si PWMCoarse=0, ir a la 2ª mitad
                                          ;de la rutina TOF
02BF 1608               BSET  TOFR,TSCR   ;Borra el Flag de Desbordamiento
                                          ; del Timer
02C1 1A08               BSET  TOIE,TSCR   ;reactiva la interrupción TOF
02C3 80     RTIInt3     RTI               ;Vuelve desde interrupción RTIF

            ;*******************************************************
03F8                    ORG   Vectors     ;Vectores de reset e Interrupción
03F8 028D               FDB   TimerInt    ;Rutina de interrupción del Timer
03FA 0200               FDB   Start       ;IRQ Externa (No usada)
03FC 0200               FDB   Start       ;Vector SWI (No usada)
03FE 0200               FDB   Start       ;Vector de Reset
```

Resumen

Un **periférico** es un trozo de hardware especializado del microcontrolador que permite a la CPU recoger la información y realizar cambios en el sistema microcontrolador.

Los puertos de E/S de propósito general se pueden programar para actuar como entradas o salidas. Cuando se configura un pin del puerto para actuar como entrada, la CPU puede leer el nivel lógico que está presente en el pin del puerto. Cuando se configura como salida, la CPU puede poner el nivel de salida del pin del puerto a un nivel lógico 1 o 0. Aunque todos los microcontroladores contienen algún puerto de propósito general de E/S como periférico, también contienen periféricos adicionales que realizan tareas más específicas.

Otros Tipos de Periféricos

Temporizadores (Timers): son periféricos que se usan para medir o generar eventos relacionados con el tiempo en un sistema microcontrolador. Los temporizadores son capaces realizar medidas de frecuencia o generando trenes de ancho de pulso variable. Los temporizadores pueden ser simples o sofisticados.

Puertos Serie: A veces los microcontroladores necesitan comunicar con periféricos externos especializados o con otro sistema microcontrolador. La comunicación normalmente se realiza con bits en serie (cada vez un bit de información). Los puertos son del tipo SCI (Interface de Comunicación Serie) y SPI (Interface serie de periféricos). La **comunicación asincrónica SCI** con otros dispositivos normalmente se usa para intercambiar datos entre dos sistemas computerizados. La **comunicación síncrona SCI** con otros dispositivos normalmente se usa para el control de dispositivos periféricos externos al microcontrolador.

Convertidor Analógico a Digital: Muchas señales que existen fuera del microcontrolador están variando continuamente la señal analógica. Un convertidor analógico a digital (**A/D**) es un periférico que se usa para convertir estas señales en un número binario que los microcontrolador pueden usar.

Convertidor Digital a Analógico: Un convertidor digital a analógico (**D/A**) realiza la función opuesta del convertidor A/D. Permite al microcontrolador convertir un número digital en un voltaje analógico proporcional o a una corriente, que se puede usar para controlar varios dispositivos de salida en un sistema microcontrolador.

EEPROM: Aunque la EEPROM es un tipo de memoria no-volátil, es considerado un periférico. La EEPROM se puede borrar el contenido y se puede volver a escribir bajo el control del programa. Existen algunos dispositivos EEPROM, como un dispositivo separado, al que puede conectarse a través de un puerto SPI.

Juego de Instrucciones

Índice

Introducción

En las definiciones se usan las nomenclaturas siguientes:

(a) Operandos

( ) = Contenidos de Registro o Posición de Memoria Mostrados entre Paréntesis
← = Está Cargado con (Lee: coger)
↑ = se Saca de la Pila
↓ = se Guarda en la Pila
• = AND Booleana
+ = Suma Aritmética (Excepto donde se usó como OR Inclusiva en la fórmula Booleana)
⊕ = OR Exclusiva Booleana
X = Multiplica
: = Encadena
- = Negado (Complemento a Dos)

(b) Registros de la CPU

ACCA = Acumulador
CCR = Registro de Código de Condición
X = Registro de Índice
PC = Contador de Programa
PCH = Contador de Programa, Orden más Alto (los 8 bits más significativos)
PCL = Contador de Programa, Orden más Bajo (los 8 bits menos significativos)
SP = Puntero de Pila

(c) Memoria y Direccionamiento

M = Una posición de memoria o datos absolutos, dependiendo del modo de direccionamiento
Rel = Desplazamiento Relativo; por ejemplo, el número complemento a dos guardado en el último byte de código de código máquina que corresponde a una instrucción de bifurcación

(d) Bits del Registro de Código de Condición (CCR)

H = Medio Acarreo, Bit 4
I = Máscara de Interrupción, Bit 3
N = Indicador de Negativo, Bit 2
Z = Indicador de Cero, Bit 1
C = Acarreo (Carry/Borrow), Bit 0

(e) Bit de Estado ANTES de la Ejecución (n = 7, 6, 5. . . 0)

An = Bit n del registro ACCA
Xn = Bit n del registro X
Mn = Bit n del registro M

(f) Bit de estado DESPUÉS de la ejecución

Rn = Bit n del Resultado (n = 7, 6, 5. . . 0)

(*g) Actividad Resumen del CCR, símbolos utilizados*

— = Bit no Afectado

0 = Bit Forzado a 0
1 = Bit Forzado a 1
↕ = Bit Puesto 0 o a 1 según los Resultados de la Operación

(h) Anotación utilizada del Código Máquina

dd = Los 8 bits más bajos de una Dirección Directa $0000-$00FF;
Byte Alto asumido para ser $0000
ee = Los 8 Bits más Altos de un Desplazamiento de 16 bits
ff = Los 8 Bits más Bajos de un Desplazamiento de 16 bits o de 8-Bits
ii = Un Byte de Datos Inmediato
hh = El Byte más Alto de los 16-Bits de una Dirección Extendida
ll = El Byte más Bajo de los 16-Bits de una Dirección Extendida
rr = Desplazamiento Relativo

(i) Anotación de forma de Fuente

(opr) = Operando; Uno o Dos Bytes que dependen del Modo de Direccionamiento
(rel) = Desplazamiento Relativo Usado en instrucciones de Bifurcación y de Manipulación de Bit

Juego de Instrucciones del MC68HC05

Las páginas siguientes contienen información detallada de todas las instrucciones del MC68HC05. Las instrucciones están colocadas por orden alfabético con el juego de instrucciones mnemónicas, para una referencia más fácil.

Operación: **ACCA ← (ACCA) + (M) + (C)**

Descripción: Suma los contenidos del bit C a la suma de los contenidos del registro ACCA y del registro M, pone el resultado en el registro ACCA.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	↕	—	↕	↕	↕

H $A3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet A3$
Se pone a 1 si había un acarreo del bit 3; de lo contrario se pone a 0.

N $R7$
Se pone a 1 si el resultado del MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet A7$
Se pone a 1 si había un acarreo del resultado del MSB; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	**Modo de Direccionamiento**	**Código Máquina**			**Ciclos**
		Opcode	**Operando(s)**		
ADC (opr)	IMM	A9	ii		2
ADC (opr)	DIR	B9	dd		3
ADC (opr)	EXT	C9	hh	ll	4
ADC ,X	IX	F9			3
ADC (opr),X	IX1	E9	ff		4
ADC (opr),X	IX2	D9	Ee	ff	5

ADD *Suma sin Acarreo* *ADD*

Operación: **ACCA ← (ACCA) + (M)**

Descripción: Suma los contenidos de M a los contenidos de ACCA y pone el resultado en ACCA.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	↕	—	↕	↕	↕

H $A3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet A3$
Se pone a 1 si había un acarreo del bit 3; de lo contrario se pone a 0.

N $R7$
Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet A7$
Se pone a 1 si había un acarreo del resultado del MSB; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
ADD (opr)	IMM	AB	ii		2
ADD (opr)	DIR	BB	dd		3
ADD (opr)	EXT	CB	hh	ll	4
ADD ,X	IX	FB			3
ADD (opr),X	IX1	EB	ff		4
ADD (opr),X	IX2	DB	ee	ff	5

AND *AND Lógico* *AND*

Operación: **ACCA ← (ACCA) · (M)**

Descripción: Realiza un AND lógico entre el contenido ACCA y M, pone el resultado en ACCA. (Cada bit del ACCA después de la operación será un AND lógico de los correspondientes bits de M y ACCA antes de la operación.)

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

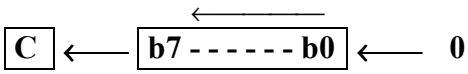
Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
AND (opr)	IMM	A4	ii		2
AND (opr)	DIR	B4	dd		3
AND (opr)	EXT	C4	hh	ll	4
AND ,X	IX	F4			3
AND (opr),X	IX1	E4	ff		4
AND (opr),X	IX2	D4	ee	ff	5

ASL Desplazamiento Aritmético a la Izquierda ASL

(Lo mismo que LSL)

Operación:



Descripción: Desplaza un lugar a la izquierda todos los bits del ACCA, X o M. El Bit 0 está cargado con un cero. El bit C en el CCR está cargado con el bit más significativo de ACCA, X o M.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N *R7*

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C b7

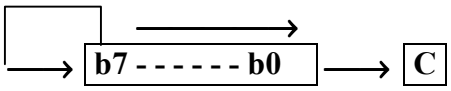
Se pone a 1 si antes del desplazamiento el valor MSB del valor desplazado era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
ASLA	INH (A)	48		3
ASLX	INH (X)	58		3
ASL (opr)	DIR	38	dd	5
ASL	IX	78		5
ASL (opr),X	IX1	68	ff	6

ASR Desplazamiento Aritmético a la Derecha ASR

Operación:



Descripción: Desplaza todos los bits un lugar a la derecha del ACCA, X o M. El Bit 7 se mantiene constante. El Bit 0 está cargado en el Bit C del CCR. Esta operación divide eficazmente un valor complemento a dos por dos sin cambiar su signo. El bit de acarreo se puede usar para redondear el resultado.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C b0

Se pone a 1 si antes del desplazamiento, el valor LSB del valor desplazado era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
ASRA	INH (A)	47		3
ASRX	INH (X)	57		3
ASR (opr)	DIR	37	dd	5
ASR ,X	IX	77		5
ASR (opr),X	IX1	67	ff	6

BCC Bifurcación si se pone a 0 el Acarreo BCC

(Lo mismo que BHS)

Operación: **PC ← (PC) + $0002 + Rel** **Si (C) = 0**

Descripción: Prueba el estado del bit C en el CCR y provoca una bifurcación si C está a 0. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BCC (rel)	REL	24	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	r ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	r < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	r ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	r > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BCLR n Pone a 0 un Bit en la Memoria BCLR n

Operación: **Mn ← 0**

Descripción: Pone a 0 el Bit n (n = 7, 6, 5. . 0) en la posición M. Todos los otros bits en M no están afectados. M puede ser cualquier posición de la RAM o dirección del registro de E/S en el área de memoria de $0000 a $00FF (por ejemplo, en el modo de direccionamiento directo se usa para especificar la dirección del operando).

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BCLR 0,(opr)	DIR (bit 0)	11	dd	5
BCLR 1,(opr)	DIR (bit 1)	13	dd	5
BCLR 2,(opr)	DIR (bit 2)	15	dd	5
BCLR 3,(opr)	DIR (bit 3)	17	dd	5
BCLR 4,(opr)	DIR (bit 4)	19	dd	5
BCLR 5,(opr)	DIR (bit 5)	1B	dd	5
BCLR 6,(opr)	DIR (bit 6)	1D	dd	5
BCLR 7,(opr)	DIR (bit 7)	1F	dd	5

BCS *Bifurcación si el Acarreo es 1* *BCS*

(Lo mismo que BLO)

Operación: **PC ← (PC) + $0002 + Rel** **Si (C) = 1**

Descripción: Prueba el estado del bit C en el CCR y provoca una bifurcación, si C está a 1. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BCS (rel)	REL	25	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BEQ *Bifurcación si es Igual* *BEQ*

Operación: **PC ← (PC) + $0002 + Rel** **Si (Z) = 1**

Descripción: Prueba el estado del bit Z en el CCR y provoca una bifurcación si Z está a 1. Siguiendo a una instrucción CMP o SUB, la instrucción BEQ causará una bifurcación si los argumentos son iguales. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BEQ (rel)	REL	27	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BHCC Bifurcación si Medio Acarreo es 0 BHCC

Operación: **PC ← (PC) + $0002 + Rel** **Si (H) = 0**

Descripción: Prueba el estado del bit H en el CCR y provoca una bifurcación si H está a 0. Esta instrucción se usa en algoritmos que involucran números BCD. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BHCC (rel)	REL	28	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BHCS Bifurcación si Medio Acarreo es 1 BHCS

Operación: **PC ← (PC) + $0002 + Rel** **Si (H) = 1**

Descripción: Prueba el estado del bit H en el CCR y provoca una bifurcación si H está a 1. Esta instrucción se usa en algoritmos que involucran números BCD. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BHCS (rel)	REL	29	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BHI Bifurcación si más es Mayor BHI

Operación: **C ← (PC) + $0002 + Rel** **Si (C) + (Z) = 0**
por ejemplo, si (ACCA) > (M) (números binarios sin signo)

Descripción: Causa una bifurcación si se pone C y Z a 0. Si la instrucción BHl se ejecuta inmediatamente después de la ejecución de una instrucción CMP o SUB, ocurrirá la bifurcación si el número binario sin signo en ACCA es mayor que el número binario sin signo en M. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BHI (rel)	REL	22	Rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BHS *Bifurcación si es Mayor o Igual* *BHS*

(Lo mismo que la instrucción BCC)

Operación: **PC ← (PC) + $0002 + Rel** **Si (C) = 0**
por ejemplo, si (ACCA) ≥ (M) (números binarios sin signo)

Descripción: Si la instrucción BHS se ejecuta inmediatamente después de la ejecución de una instrucción CMP o SUB, la bifurcación ocurrirá si el número binario sin signo en ACCA era mayor o igual al número binario sin signo en M. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BHS (rel)	REL	24	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BIH Bifurcación si el Pin de Interrupción está en nivel Alto BIH

Operación: **PC ← (PC) + $0002 + Rel** **Si IRQ = 1**

Descripción: Prueba el estado del pin de interrupción externa y provoca una bifurcación si el pin está en nivel alto. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BIH (rel)	REL	2F	Rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BIL Bifurcación si el Pin de Interrupción está en nivel Bajo BIL

Operación: **PC ← (PC) + $0002 + Rel** **Si IRQ = 0**

Descripción: Prueba el estado del pin de interrupción externa y provoca una bifurcación si el pin está en nivel bajo. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BIL (rel)	REL	2E	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BIT Bit de Prueba de la Memoria con el Acumulador BIT

Operación: **(ACCA) · (M)**

Descripción: Realiza una comparación lógica AND de los contenidos de ACCA y M, y modifica de acuerdo el código de condición. No se alteran los contenidos de ACCA ni de M. (Cada bit del resultado AND lógico, serán los bits correspondientes de ACCA y M).

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
BIT (opr)	IMM	A5	ii		2
BIT (opr)	DIR	B5	dd		3
BIT (opr)	EXT	C5	hh	ll	4
BIT ,X	IX	F5			3
BIT (opr),X	IX1	E5	ff		4
BIT (opr),X	IX2	D5	Ee	ff	5

BLO *Bifurcación si es más Menor* *BLO*

(Lo mismo que la instrucción BCS)

Operación: **PC ← (PC) + $0002 + Rel** **Si (C) = 1**
por ejemplo, si (ACCA) < (M) (números binarios sin signo)

Descripción: Si la instrucción BLO se ejecuta inmediatamente después de la ejecución de una instrucción CMP o SUB, la bifurcación ocurrirá si el número binario sin signo en ACCA era menor del número binario sin signo en M. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BLO (rel)	REL	25	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/ BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/ BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHC C	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BLS Bifurcación si es Menor o Igual BLS

Operación: **PC ← (PC) + $0002 + Rel** **Si [(C) + (Z)] = 1**
por ejemplo, si (ACCA) ≤ (M) (números binarios sin signo)

Descripción: Causa una bifurcación si C o Z se pone a 1. Si la instrucción BLS se ejecuta inmediatamente después de la ejecución de una instrucción CMP o SUB, la bifurcación ocurrirá si el número binario sin signo en ACCA fue menor o igual al número binario sin signo en M. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BLS (rel)	REL	23	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BMC Bifurcación si la Máscara de Interrupción es 0 BMC

Operación: **PC ← (PC) + $0002 + Rel** **Si I = 0**

Descripción: Prueba el estado del bit I en el CCR y causa una bifurcación si I es cero (por ejemplo, si se habilitan las interrupciones). Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BMC (rel)	REL	2C	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BMI Bifurcación si es Menor BMI

Operación: **PC ← (PC) + $0002 + Rel** **Si (N) = 1**

Descripción: Prueba que el estado del bit N en el CCR y causa una bifurcación si N es 1. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BMI (rel)	REL	2B	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BMS Bifurcación si la Máscara de Interrupción es 1 BMS

Operación: **PC ← (PC) + $0002 + Rel** **Si (I) = 1**

Descripción: Prueba el estado del bit I en el CCR y causa una bifurcación si I es 1 (por ejemplo, si las interrupciones son inválidas). Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BMS (rel)	REL	2D	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	R ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BNE Bifurcación si no es Igual BNE

Operación: **PC ← (PC) + $0002 + Rel** **Si (Z) = 0**

Descripción: Prueba el estado del bit Z en el CCR y provoca una bifurcación si Z está a 0. Siguiendo una instrucción de comparación o substracción, la instrucción BEQ provocará una bifurcación si los argumentos no serán iguales. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BNE (rel)	REL	26	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BPL Bifurcación si es Positivo BPL

Operación: **PC ← (PC) + $0002 + Rel** **Si (N) = 0**

Descripción: Prueba el estado del bit N en el CCR y provoca una bifurcación si N está a 0. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BPL (rel)	REL	2A	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2ª	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BRA Bifurcación Incondicional BRA

Operación: **PC ← (PC) + $0002 + Rel**

Descripción: Bifurcación incondicional a la dirección dada por la fórmula anterior, en qué 'Rel' es el desplazamiento relativo guardado como un número complemento a dos en el último byte de código máquina correspondiente a la instrucción de bifurcación. PC es la dirección del 'opcode' para la instrucción bifurcación.

El programa fuente especifica el destino de cualquier instrucción de bifurcación por su dirección absoluta o como un valor numérico o como un símbolo o expresión que puede ser evaluada numéricamente por el ensamblador. El ensamblador calcula la dirección relativa 'Rel' de la dirección absoluta y el valor actual de la posición del contador.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BRA (rel)	REL	20	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BRCLR n Bifurcación si el Bit n es Cero BRCLR n

Operación: **PC ← (PC) + $0003 + Rel** **Si el bit n de M = 0**

Descripción: Prueba el bit n (N = 7, 6, 5...0) de la posición M y bifurca si el bit se pone a 0, M puede ser una posición de RAM o una dirección del registro de E/S en el área de memoria $0000 a $00FF (por ejemplo, modo de direccionamiento directo si está usado para especificar la dirección del operando).

El bit C es 1 para el estado de prueba de bit. Cuando se usa junto con una apropiada instrucción de rotación, BRCLR n mantiene un método fácil realizando sesiones de serie a paralelo.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	↕

C se pone a 1 si Mn = 1; por otra parte es cero

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
BRCLR 0,(opr)	DIR (bit 0)	01	dd	rr	5
BRCLR 1,(opr)	DIR (bit 1)	03	dd	rr	5
BRCLR 2,(opr)	DIR (bit 2)	05	dd	rr	5
BRCLR 3,(opr)	DIR (bit 3)	07	dd	rr	5
BRCLR 4,(opr)	DIR (bit 4)	09	dd	rr	5
BRCLR 5,(opr)	DIR (bit 5)	0B	dd	rr	5
BRCLR 6,(opr)	DIR (bit 6)	0D	dd	rr	5
BRCLR 7,(opr)	DIR (bit 7)	0F	dd	rr	5

BRN *Nunca Bifurcación* *BRN*

Operación: **PC ← (PC) + $0002**

Descripción: Nunca hace bifurcación. En efecto, esta instrucción puede ser considerada como 2-bytes NOP (no operación) requiriendo tres ciclos para su ejecución. Esta inclusión en el juego de instrucciones es para mantener un complemento de la instrucción BRA. La instrucción es útil durante el programa de depuración para negar el efecto de otra instrucción de bifurcación sin perturbar el byte de desplazamiento.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BRN (rel)	REL	21	rr	3

Esta tabla es un resumen de todas las instrucciones de bifurcación.

Prueba	Boole	Mnemónico	Opcode	Complementario		Bifurcación	Comentario
r > m	C + Z = 0	BHI	22	R ≤ m	BLS	23	Sin signo
r ≥ m	C = 0	BHS/BCC	24	R < m	BLO/BCS	25	Sin signo
r = m	Z = 1	BEQ	27	R ≠ m	BNE	26	Sin signo
r ≤ m	C + Z = 1	BLS	23	R > m	BHI	22	Sin signo
r < m	C = 1	BLO/BCS	25	R ≥ m	BHS/BCC	24	Sin signo
Acarreo	C = 1	BCS	25	Sin Acarreo	BCC	24	Simple
r = 0	Z = 1	BEQ	27	r ≠ 0	BNE	26	Simple
Negativo	N = 1	BMI	2B	Más	BPL	2A	Simple
I Enmascarable	I = 1	BMS	2D	I Enmascarable = 0	BMC	2C	Simple
Medio Acarreo	H = 1	BHCS	29	No Medio Acarreo	BHCC	28	Simple
IRQ Pin Alto	—	BIH	2F	IRQ Bajo	BIL	2E	Simple
Siempre	—	BRA	20	Nunca	BRN	21	Incondicional

r = registro (ACCA o X); m = operando memoria

BRSET n *Bifurcación si el Bit n es 1* *BRSET*

Operación: **PC ← (PC) + $0003 + Rel** **Si el Bit n de M = 1**

Descripción: Prueba el Bit n (n = 7, 6, 5, 0) de la posición M y bifurca si el bit está a 1. M puede ser cualquier posición de la RAM o dirección del registro de E/S en el área de memoria $0000 a $00FF (por ejemplo, el modo de direccionamiento directo se usa para especificar la dirección del operando). El Bit C se pone al estado del bit probado. Cuando se usó junto con una apropiada instrucción de rotación, BRSET n proporciona un método fácil para realizar sesiones de serie a paralelo.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	↕

C se pone a 1 si Mn = 1; por el contrario es cero

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
BRSET 0,(opr)	DIR (bit 0)	00	dd	rr	5
BRSET 1,(opr)	DIR (bit 1)	02	dd	rr	5
BRSET 2,(opr)	DIR (bit 2)	04	dd	rr	5
BRSET 3,(opr)	DIR (bit 3)	06	dd	rr	5
BRSET 4,(opr)	DIR (bit 4)	08	dd	rr	5
BRSET 5,(opr)	DIR (bit 5)	0C	dd	rr	5
BRSET 6,(opr)	DIR (bit 6)	0E	dd	rr	5
BRCLR 7,(opr)	DIR (bit 7)	0F	dd	rr	5

BSET n Pone a 1 el Bit en la Memoria BSET n

Operación: **Mn ← 1**

Descripción: Pone el Bit n (n =7,6,5 ...0) en la posición M. Todos los otros bits en M son afectados. M puede ser cualquier posición de la RAM o dirección del registro de E/S en el área de memoria de $0000 a $00FF (por ejemplo, el modo de direccionamiento directo se usa para especificar la dirección del operando).

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BSET 0,(opr)	DIR (bit 0)	10	dd	5
BSET 1,(opr)	DIR (bit 1)	12	dd	5
BSET 2,(opr)	DIR (bit 2)	14	dd	5
BSET 3,(opr)	DIR (bit 3)	16	dd	5
BSET 4,(opr)	DIR (bit 4)	18	dd	5
BSET 5,(opr)	DIR (bit 5)	1A	dd	5
BSET 6,(opr)	DIR (bit 6)	1C	dd	5
BCLR 7,(opr)	DIR (bit 7)	1E	dd	5

BSR Bifurcación a Subrutina BSR

Operación

PC ← (PC) + $0002	Avanza el PC para volVéase a la dirección
↓ (PCL); SP ← (SP) – $0001	Guarda el orden más bajo hacia la pila
↓ (PCL); SP ← (SP) – $0001	Guarda el orden más alto hacia la pila
PC ← (PC) + Rel	Carga el PC con la dirección de inicio de la subrutina pedida

Descripción: El contador de programa es incrementado a través de dos direcciones del 'opcode', por ejemplo, apunta al 'opcode' de la siguiente instrucción que quiere ser la dirección de retorno. El byte menos significativo de los contenidos de contador de programa (dirección de retorno de orden más bajo) se pone en la pila. El puntero de pila entonces es decrementado por uno. El byte más significativo de los contenidos del contador de programa (dirección de retorno de orden más alto) se pone en la pila. El puntero de pila entonces es decrementado por uno. Entonces ocurre una bifurcación a la posición especificada por el desplazamiento de la bifurcación. Véase la instrucción BRA para más detalles de la ejecución de la bifurcación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
BSR (rel)	REL	AD	rr	6

CLC Pone a Cero el Bit de Acarreo CLC

Operación: **Bit C ← 0**

Descripción: Pone a cero el bit C en el CCR. La instrucción CLC puede ser usada para preparar el bit C antes de una instrucción de desplazamiento o rotación que involucre al Bit C.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	0

C = Se pone a 0

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
CLC	INH	98		2

CLI Pone a Cero el Bit de Máscara de Interrupción CLI

Operación: **Bit I ← 0**

Descripción: Pone a cero el bit de máscara de interrupción en el CCR. Cuando el bit I se pone a cero, se habilitan las interrupciones. Hay un retraso de un ciclo de E-reloj en el mecanismo de poner a cero para el bit I, por si las interrupciones fueron previamente deshabilitadas, la siguiente instrucción después de una instrucción CLI siempre se ejecutará, aun cuando había una interrupción anterior pendiente a la ejecución de la instrucción CLI.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	0	—	—	—

I se pone a 0

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
CLI	INH	9A		2

CLR *Pone a Cero* *CLR*

Operación: **ACCA ← $00** **o:** **M ← $00** **o:** **X ← $00**

Descripción: Los contenidos de ACCA, M o X, se reemplazan con Ceros.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	0	1	—

I se pone a 0
Z se pone a 1

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
CLRA	INH (A)	4F		3
CLRX	INH (X)	5F		3
CLR (opr)	DIR	3F	dd	5
CLR ,X	IX	7F		5
CLR (opr),X	IX1	6F	ff	6

CMP Compara el Acumulador con la Memoria CMP

Operación: **(ACCA) – (M)**

Descripción: Compara los contenidos de ACCA con los contenidos de M y pone a 1 el código de condición que se puede usar para la bifurcación condicional aritmética y lógica. Los contenidos de ACCA y de M son inalterados.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet \overline{A7}$

Se pone a 1 si el valor absoluto de los contenidos de memoria es mayor que el valor absoluto del acumulador; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
CMP (opr)	IMM	A1	ii		2
CMP (opr)	DIR	B1	dd		3
CMP (opr)	EXT	C1	hh	ll	4
CMP ,X	IX	F1			3
CMP (opr),X	IX1	E1	ff		4
CMP (opr),X	IX2	D1	ee	ff	5

COM *Complemento* *COM*

Operación: **ACCA ← (ACCA) = $FF – (ACCA)** **o:** **M ← (M) = $FF – (M)**
o: **X ← X = $FF – (X)**

Descripción: Reemplaza los contenidos de ACCA, X o M con su complemento a uno. (Cada bit de los contenidos de ACCA, X o M se reemplazan con el complemento de ese Bit.)

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	1

N $R7$
Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

C
Se pone a 1

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
COMA	INH (A)	43		3
COMX	INH (X)	53		3
COM (opr)	DIR	33	dd	5
COM ,X	IX	73		5
COM (opr),X	IX1	63	ff	6

CPX Compara el Registro de Índice con la Memoria CPX

Operación: **(X) – (M)**

Descripción: Compara los contenidos del registro de índice con los contenidos de la memoria y pone a 1 el código de condición que se pueden usar para la bifurcación aritmética y lógica. Los contenidos de ACCA y M están inalterados.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C $\overline{IX7} \bullet M7 + M7 \bullet \mathrm{R7} + R7 \bullet \overline{IX7}$

Se pone a 1 si el valor absoluto de los contenidos de memoria es mayor que el valor absoluto del registro de índice; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
CPX (opr)	IMM	A3	ii		2
CPX (opr)	DIR	B3	dd		3
CPX (opr)	EXT	C3	hh	ll	4
CPX ,X	IX	F3			3
CPX (opr),X	IX1	E3	ff		4
CPX (opr),X	IX2	D3	ee	ff	5

DEC *Decremento* *DEC*

Operación: **ACCA ← (ACCA) – $01** **o** **M ← (M) – $01** **o** **X ← (X) - $01**

Descripción: Substrae uno de los contenidos de ACCA, X o M. Los bits N y Z en el CCR se ponen a 1 o 0 según el resultado de esta operación. El bit C en el CCR no es afectado; por consiguiente, las únicas instrucciones de bifurcación que son útiles siguiendo a una instrucción DEC son BEQ, BNE, BPL y BMI.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	**Modo de Direccionamiento**	**Código Máquina**		**Ciclos**
		Opcode	**Operando(s)**	
DECA	IMM	4A		3
DECX	DIR	5A		3
DECA (opr)	EXT	3A	dd	5
DEC ,X	IX	7A		5
DEC (opr),X	IX1	6A	ff	6

DEX es reconocido por el ensamblador com una equivalencia de DECX

EOR OR-Exclusiva de la Memoria con el Acumulador EOR

Operación: **ACCA ← (ACCA) ⊕ (M)**

Descripción: Realiza una OR-Exclusiva lógico entre los contenidos de ACCA y M, pone el resultado en ACCA. (Cada bit de ACCA después de una operación será una OR-Exclusiva lógico de los bits correspondientes de M y ACCA antes de la operación.)

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
EOR (opr)	IMM	A8	ii		2
EOR (opr)	DIR	B8	dd		3
EOR (opr)	EXT	C8	hh	ll	4
EOR ,X	IX	F8			3
EOR (opr),X	IX1	E8	ff		4
EOR (opr),X	IX2	D8	ee	ff	5

INC *Incrementa* *INC*

Operación: **ACCA ← (ACCA) + $01 o M ← (M) + $01 o X ← (X) + $01**

Descripción: Suma uno a los contenidos de ACCA, X o M. Los bits N y Z en el CCR son puestos a 1 o a 0 según los resultados de esta operación. El bit C en el CCR no es afectado; por consiguiente, las únicas instrucciones de bifurcación que son útiles a instrucción INC son BEQ, BNE, BPL y BMI.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
INCA	IMM	4C		3
INCX	DIR	5C		3
INC (opr)	EXT	3C	dd	5
INC ,X	IX	7C		5
INC (opr),X	IX1	6C	ff	6

INX es reconocido por el ensamblador con una equivalencia a INCX

JMP *Salto* *JMP*

Operación: **PC ← Dirección Efectiva**

Descripción: Ocurre un salto a la instrucción guardada a la dirección efectiva. La dirección efectiva se obtiene de acuerdo con las reglas del modo de direccionamiento Extendido, Directo o Indexado.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
JMP (opr)	DIR	BC	dd		2
JMP (opr)	EXT	CC	hh	ll	3
JMP ,X	IX	FC			2
JMP (opr),X	IX1	EC	ff		3
JMP (opr),X	IX2	DC	ee	ff	4

JSR *Salto a Subrutina* *JSR*

Operación:

PC ← (PC) + n	n = 1, 2, 3 dependiendo del modo de direccionamiento
↓ (PCL); SP ← (SP) – $0001	Pone la parte baja de la dirección de retorno a la pila
↓ (PCL); SP ← (SP) – $0001	Pone la parte alta de la dirección de retorno a la pila
PC ← Dirección Efectiva	Carga el PC con dirección de inicio de la subrutina pedida

Descripción: El contador de programa es incrementado por *n* que apunta al opcode de la instrucción que sigue a la instrucción JSR (*n* = 1, 2 o 3 dependiendo del modo de direccionamiento). Entonces se Pone el PC hacia la pila, ocho bits de una vez, primero el byte menos significativo. Los bits sin usar en el contador de programa, el byte alto, se guarda como unos en la pila. El puntero de pila apunta a la siguiente posición vacía en la pila. Ocurre un salto a la instrucción guardada a la dirección efectiva. La dirección efectiva se obtiene según las reglas de los modos de direccionamiento Extendido, Directo o Indexado.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
JSR (opr)	DIR	BD	dd		5
JSR (opr)	EXT	CD	hh	ll	6
JSR ,X	IX	FD			5
JSR (opr),X	IX1	ED	ff		6
JSR (opr),X	IX2	DD	ee	Ff	7

LDA Carga el Acumulador desde la Memoria LDA

Operación: **ACCA ← (M)**

Descripción: Carga los contenidos de la memoria en el acumulador. El código de condición se pone a 1 de acuerdo los datos.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
LDA (opr)	IMM	A6	ii		2
LDA (opr)	DIR	B6	dd		3
LDA (opr)	EXT	C6	hh	ll	4
LDA ,X	IX	F6			3
LDA (opr),X	IX1	E6	ff		4
LDA (opr),X	IX2	D6	ee	ff	5

LDX Carga el Registro de Índice desde la Memoria LDX

Operación: **X ← (M)**

Descripción: Carga los contenidos de la posición de memoria especificada en el registro de índice. Los códigos de condición son puestos a 1 de acuerdo a los datos.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

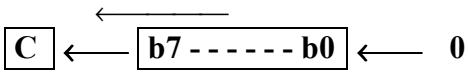
Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	**Modo de Direccionamiento**	**Código Máquina**			**Ciclos**
		Opcode	**Operando(s)**		
LDX (opr)	IMM	AE	ii		2
LDX (opr)	DIR	BE	dd		3
LDX (opr)	EXT	CE	hh	ll	4
LDX ,X	IX	FE			3
LDX (opr),X	IX1	EE	ff		4
LDX (opr),X	IX2	DE	ee	ff	5

LSL Desplazamiento Lógico a la Izquierda LSL

(lo mismo que la instrucción ASL)

Operación:



Descripción: Desplaza todos los bits de ACCA, X o M un lugar a la izquierda. El bit 0 está cargado con 0. El bit C en el CCR está cargado desde el bit más significativo de ACCA, X o M.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N *R7*

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C b7

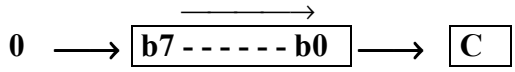
Se pone a 1 si antes del desplazamiento el valor MSB del valor desplazado era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
LSLA	INH (A)	48		3
LSLX	INH (X)	58		3
LSL (opr)	DIR	38	dd	5
LSL	IX	78		5
LSL (opr),X	IX1	68	ff	6

LSR Desplazamiento Lógico a la Derecha LSR

Operación



Descripción: Cambia todos los bits de ACCA, X o M un lugar a la derecha. El bit 7 está cargado con 0. El bit 0 se desplaza al bit C.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	0	↕	↕

N Es 0

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se ponen a 0; de lo contrario se pone a 0.

C b0

Se pone a 1 si antes del desplazamiento, el valor LSB de ACCA, X o M era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
LSRA	INH (A)	44		3
LSRX	INH (X)	54		3
LSR (opr)	DIR	34	dd	5
LSR ,X	IX	74		5
LSR (opr),X	IX1	64	ff	6

MUL Multiplicación Sin Signo MUL

Operación: **X : A ← X x A**

Descripción: Multiplica los ocho bits del registro de índice por los ocho bits del acumulador para obtener un número de 16-bits sin signo, encadenando el registro de índice y el acumulador. Después de la operación, X contiene los 8 bits más altos del resultado de 16-bits.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	0	—	—	—	0

H Se pone a 0

C Se pone a 0

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
MUL	INH	42		3

NEG *Negado* *NEG*

Operación: **ACCA ← – (ACCA); o: X ←— (X); o: M ← – (M)**

Descripción: Reemplaza los contenidos de ACCA, X o M con su complemento a dos. El valor $80 queda inalterado.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N *R7*

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

C R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0

Se pone a 1 si hay un acarreo por substracción implícita de 0; se pone a cero por otro lado. El bit C se pondrá a 1 en todos los casos excepto cuando los contenidos de ACCA, X o M (anterior a la operación NEG) es $00.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
NEGA	INH (A)	40		3
NEGX	INH (X)	50		3
NEG (opr)	DIR	30	dd	5
NEG ,X	IX	70		5
NEG (opr),X	IX1	60	ff	6

NOP *No Operación* *NOP*

Descripción: Ésta es una instrucción de un solo byte que causa al contador del programa que sea incrementado. Ningún otro registro es afectado.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
NOP	INH	9D		2

ORA *OR-Inclusiva* *ORA*

Operación: **ACCA ← (ACCA) + (M)**

Descripción: Realiza una OR-Inclusiva lógico entre los contenidos de ACCA y M, pone el resultado en ACCA. Cada bit de ACCA, después de la operación, será una OR-Inclusiva lógico de los bits correspondientes de M y ACCA antes de la operación.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

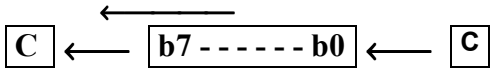
Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
ORA (opr)	IMM	AA	ii		2
ORA (opr)	DIR	BA	dd		3
ORA (opr)	EXT	CA	hh	ll	4
ORA ,X	IX	FA			3
ORA (opr),X	IX1	EA	ff		4
ORA (opr),X	IX2	DA	ee	ff	5

ROL Rotación a la Izquierda por Acarreo ROL

Operación:



Descripción: Desplaza todos los bits de ACCA, X o M un lugar a la izquierda. El bit 0 está cargado desde el bit C. El bit C está cargado desde MSB de ACCA, X, o M. Las instrucciones de rotación incluyen el bit de acarreo para permitir la extensión de las operaciones de desplazamiento y de rotación a los múltiples bytes. Por ejemplo, para desplazar un valor de 24-bits un bit a la derecha, se podría usar la sucesión {ASL BAJO, ROL MEDIO, ROL ALTO} donde BAJO, MEDIO y ALTO, se refiere a la parte baja, media y alta del byte de valor de 24-bits, respectivamente.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

C b7

Se pone a 1 si antes de la rotación, el MSB de ACCA o M era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
ROLA	INH (A)	49		3
ROLX	INH (X)	59		3
ROL (opr)	DIR	39	dd	5
ROL ,X	IX	79		5
ROL (opr),X	IX1	69	ff	6

ROR Rotación a la Derecha por Acarreo ROR

Operación:

Descripción: Desplaza todos los bits de ACCA, X o M un lugar a la derecha. El bit 7 está cargado desde el bit C. Las operaciones de rotación incluyen el bit de acarreo para permitir la extensión de las operaciones de desplazamiento y de rotación a los múltiples bytes. Por ejemplo, para desplazar un valor de 24-bits a la derecha un bit, se puede usar la sucesión {LSR ALTO, ROR MEDIO, ROR BAJO} donde ALTO, MEDIO y BAJO se refiere a la parte alta, media y baja del byte de valor de 24-bits, respectivamente.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado se pone a 0; de lo contrario se pone a 0.

C b0

Se pone a 1 si antes de la rotación, el LSB de ACCA o M era 1; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
RORA	INH (A)	46		3
RORX	INH (X)	56		3
ROR (opr)	DIR	36	dd	5
ROR ,X	IX	76		5
ROR (opr),X	IX1	66	ff	6

RSP Reset del Puntero de Pila RSP

Operación: **SP ← $00FF**

Descripción: Reset al puntero de pila en la parte alta de la pila.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
RSP	INH	9C		2

RTI Retorno de la Interrupción RTI

Operación:
SP ← (SP) + $0001; ↑ CCR Restablece el CCR desde la pila
SP ← (SP) + $0001; ↑ ACCA Restablece el ACCA desde la pila
SP ← (SP) + $0001; ↑ X Restablece el X desde la pila
SP ← (SP) + $0001; ↑ PCH Restablece el PCH desde la pila
SP ← (SP) + $0001; ↑ PCL Restablece el PCL desde la pila

Descripción: Se restablece el código de condición, el acumulador, el registro del índice y el contador de programa, que previamente al estado guardado en la pila. Se restablece a bit 1 si el bit correspondiente guardado en la pila es 0.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	↕	↕	↕	↕	↕

Se pone a 1 o 0 según el byte sacado desde la pila.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
RTI	INH	80		9

RTS — *Retorno de Subrutina* — *RTS*

Operación: **SP ← (SP) + $0001; ↑ PCH** Restablece PCH desde la pila
SP ← (SP) + $0001; ↑ PCL Restablece PCL desde la pila

Descripción: El puntero de pila es incrementado por uno. El contenido del byte de la memoria que se apunta por el puntero de pila está cargada en el byte de la parte alta del contador del programa. El puntero de pila es de nuevo incrementado por uno. El contenido del byte de la memoria a la dirección ahora contenida en el puntero de pila está cargado en los 8 bits de la parte baja del contador de programa.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
RTS	INH	81		6

SBC Subtracción con Acarreo SBC

Operación: **ACCA ← (ACCA) – (M) – (C)**

Descripción: Substrae los contenidos de M y C de los contenidos de ACCA, pone el resultado en ACCA.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado del MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

C $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$

Se pone a 1 si el valor absoluto de los contenidos de la memoria más el acarreo anterior, es más grande que el valor absoluto del acumulador; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
SBC (opr)	IMM	A2	ii		2
SBC (opr)	DIR	B2	dd		3
SBC (opr)	EXT	C2	hh	ll	4
SBC ,X	IX	F2			3
SBC (opr),X	IX1	E2	ff		4
SBC (opr),X	IX2	D2	ee	ff	5

SEC Pone a 1 el bit de Acarreo SEC

Operación: **bit C ← 1**

Descripción: Pone a 1 el bit C en el CCR. La instrucción SEC se puede usar para preparar el bit C antes de una instrucción de desplazamiento o de rotación que involucre el bit C.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	1

C Se pone a 1

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
SEC	INH	99		2

SEI Pone a 1 el Bit de Máscara de Interrupción SEI

Operación: **bit I ← 1**

Descripción: Pone a 1 el bit de máscara de interrupción en el CCR. El microprocesador se inhibe del servicio de interrupciones mientras que el bit I es 1.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	1	—	—	—

I Se pone a 1

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
SEI	INH	9B		2

STA Guarda el Acumulador en la Memoria STA

Operación: **M ← (ACCA)**

Descripción: Guarda los contenidos de ACCA en la memoria. Los contenidos de ACCA permanecen inalterados.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	—

N *A7*

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{A7} \bullet \overline{A6} \bullet \overline{A5} \bullet \overline{A4} \bullet \overline{A3} \bullet \overline{A2} \bullet \overline{A1} \bullet \overline{A0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
STA (opr)	DIR	B7	dd		4
STA (opr)	EXT	C7	hh	ll	5
STA ,X	IX	F7			4
STA (opr),X	IX1	E7	ff		5
STA (opr),X	IX2	D7	ee	ff	6

STOP Habilita la $\overline{\text{IRQ}}$ y Para el Oscilador STOP

Descripción: Reduce el consumo de potencia eliminando toda disipación de potencia dinámica. Esto resulta por: 1) Se ponen a 0 los 'prescalers' del temporizador, 2) Deshabilita las interrupciones del temporizador, 3) Pone a 0 el 'flag' de interrupción del temporizador, 4) Habilita la petición de interrupción externa y 5) Inhibe el oscilador.

Cuando un $\overline{\text{RESET}}$ o la entrada de $\overline{\text{IRQ}}$ se pone en estado bajo, se habilita el oscilador, se inicia un retardo de 1920 ciclos de reloj del procesador, permitiendo que el oscilador se estabilice, se saca el vector de petición de interrupción o el vector de reset y se ejecuta la rutina de servicio, dependiendo del signo que fue aplicado.

Se habilitan interrupciones externas siguiendo el comando STOP.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	0	—	—	—

I Se pone a 0

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
STOP	INH	8E		2

STX Guarda el Registro de Índice X en la Memoria STX

Operación: **M ← (X)**

Descripción: Guarda los contenidos de X en la memoria. Los contenidos de X permanecen inalterados.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—			—

N $X7$

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z $\overline{X7} \bullet \overline{X6} \bullet \overline{X5} \bullet \overline{X4} \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
STX (opr)	DIR	BF	dd		4
STX (opr)	EXT	CF	hh	ll	5
STX ,X	IX	FF			4
STX (opr),X	IX1	EF	ff		5
STX (opr),X	IX2	DF	ee	ff	6

SUB Substracción SUB

Operación: **ACCA ← (ACCA)–(M)**

Descripción: Substrae los contenidos de la M de los contenidos del ACCA y pone el resultado en el ACCA.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	↕	↕	↕

N $R7$

Se pone a 1 si el resultado del MSB es 1; de lo contrario se pone a 0.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si todos los bits del resultado están a 0; de lo contrario se pone a 0.

C $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet A7$

El bit C (‘flag’ de acarreo) en el registro de código de condición se pone a 1 si el valor absoluto de los contenidos de la memoria es más mayor que el valor absoluto del acumulador; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina			Ciclos
		Opcode	Operando(s)		
SUB (opr)	IMM	A0	ii		2
SUB (opr)	DIR	B0	dd		3
SUB (opr)	EXT	C0	hh	ll	4
SUB ,X	IX	F0			3
SUB (opr),X	IX1	E0	ff		4
SUB (opr),X	IX2	D0	ee	ff	5

SWI Interrupción por Software SWI

Operación:	**PC ← (PC) + $0001**	Avanza el PC para devolver la dirección
	↓ (PCL); SP ← (SP) – $0001	Pone la parte baja de la dirección de retorno hacia la pila
	↓ (PCH); SP ← (SP) – $0001	Pone la parte alta de la dirección de retorno hacia la pila
	↓ (X); SP ← (SP) – $0001	Pone el registro de índice hacia la pila
	↓ (ACCA); SP ← (SP) – $0001	Pone el acumulador hacia la pila
	↓ (CCR); SP ← (SP) – $0001	Pone el CCR hacia la pila
	Bit I ← 1	
	PCH ← ($xFFC)	Saca el vector (x=1o 3 dependiendo del dispositivo HC05)
	PCL ← ($xFFD)	

Descripción: El contador de programa es incrementado por uno. El contador de programa, el registro de índice y el acumulador se Ponen hacia la pila. Los bits del CCR se Ponen hacia la pila, con los bits H, I, N, Z y C que van a las posiciones de los bits de 4–0 y los bits de las posiciones 7, 6 y 5 contienen unos. El puntero de pila es decrementado en uno, después de cada byte de datos se guarda en la pila. Entonces el bit de máscara de interrupción es 1. El contador de programa es cargado con la dirección guardada en el vector de SWI (localizado en las posiciones de memoria n–0002 y n–0003, donde 'n' es la dirección que corresponde a un estado alto de todas las líneas del bus de direcciones). La dirección del vector de SWI se puede expresar como $xFFC:$xFFD, donde 'x' es 1 o 3 dependiendo del dispositivo MC68HC05 usado. Esta instrucción no es enmascarable por el bit I.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	1	—	—	—

I Se pone a 1

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
SWI	INH	83		10

TAX Transfiere el Acumulador al Registro de Índice TAX

Operación: **X ← (ACCA)**

Descripción: Carga el registro de índice con los contenidos del acumulador. Los contenidos del acumulador no se alteran.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
TAX	INH	97		2

TST Prueba para Negativo o Cero TST

Operación: **(ACCA) – $00** **o:** **(X) – $00** **o:** **(M) – $00**

Descripción: Pone a 1 los bits N y Z del código de condición, según los contenidos del ACCA, X o M. Los contenidos del ACCA, X y M no se alteran.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—			—

N $M7$

Se pone a 1 si el resultado del MSB es 1; de lo contrario se pone a 0.

Z $\overline{M7} \bullet \overline{M6} \bullet \overline{M5} \bullet \overline{M4} \bullet \overline{M3} \bullet \overline{M2} \bullet \overline{M1} \bullet \overline{M0}$

Se pone a 1 si el contenido del ACCA, X o M es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
TSTA	INH (A)	4D		3
TSTX	INH (X)	5D		3
TST (opr)	DIR	3D	dd	4
TST ,X	IX	7D		4
TST (opr),X	IX1	6D	ff	5

TXA Transfiere el Registro de Índice al Acumulador TXA

Operación: **ACCA ← (X)**

Descripción: Carga el acumulador con el contenido del registro de índice. No se altera el contenido del registro de índice.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	—	—	—	—

Ningún bit es afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
TXA	INH	9F		2

WAIT Habilita la Interrupción, Para el Procesador WAIT

Descripción: Reduce el consumo de potencia, eliminando la disipación de potencia dinámica. El temporizador, el prescaler del temporizador y los periféricos internos continúan operando porque ellos son fuentes potenciales de una interrupción. La instrucción WAIT provoca la habilitación de las interrupciones poniendo a 0 el bit I en el CCR y se paran los relojes de los circuitos del procesador.

Pueden habilitarse las interrupciones de los periféricos internos o pueden desactivarse por bits de control local anteriores a la ejecución de la instrucción WAIT.

Cuando un RESET o la entrada de IRQ se pone a un nivel bajo o cuando cualquier sistema interno hace una petición del servicio de interrupción, se habilitan los relojes del procesador y se procesa el reset, la IRQ u otra petición de servicio de interrupción.

Códigos de condición y Formula Boleana

			H	I	N	Z	C
1	1	1	—	0	—	—	—

I Se pone a 0

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		Ciclos
		Opcode	Operando(s)	
WAIT	INH	8F		2

Tablas de Referencia

Índice

Conversión de Hexadecimal a ASCII

El Código estándar Americano para el Intercambio de Información (ASCII) mantiene una norma ampliamente aceptada para codificar información alfanumérica en números binarios. El código original se diseñó con 7-bits más un **bit de paridad**. Como que la mayoría de las computadoras modernas trabajan mejor con valores de 8-bits, el código se ha adaptado ligeramente para que se pueda expresar como valores de 8-bits. Los siete bits de orden más bajo corresponden al código ASCII original y el octavo bit es 0.

Los primeros 32 códigos contienen códigos de control, por ejemplo: un retorno de carro. Muchos de éstos códigos especiales fueron utilizados para transmisiones de los viejos teletipos ya en desuso.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$16	SYN	$2C	, (comma)	$42	B	$58	X	$6E	n
$01	SOH	$17	ETB	$2D	- (dash)	$43	C	$59	Y	$6F	o
$02	STX	$18	CAN	$2E	. (period)	$44	D	$5A	Z	$70	p
$03	ETX	$19	EM	$2F	/	$45	E	$5B	[	$71	q
$04	EOT	$1ª	SUB	$30	0	$46	F	$5C	\	$72	r
$05	ENQ	$1B	ESC (escape)	$31	1	$47	G	$5D	]	$73	s
$06	ACK	$1C	FS	$32	2	$48	H	$5E	^	$74	t
$07	BEL (beep)	$1D	GS	$33	3	$49	I	$5F	_ (under)	$75	u
$08	BS (back sp)	$1E	RS	$34	4	$4A	J	$60	` (grave)	$76	v
$09	HT (tab)	$1F	US	$35	5	$4B	K	$61	a	$77	w
$0A	LF (linefeed)	$20	SP (space)	$36	6	$4C	L	$62	b	$78	x
$0B	VT	$21	!	$37	7	$4D	M	$63	c	$79	y
$0C	FF	$22	“	$38	8	$4E	N	$64	d	$7A	z
$0D	CR (return)	$23	#	$39	9	$4F	O	$65	e	$7B	{
$0E	SO	$24	$	$3ª	:	$50	P	$66	f	$7C	|
$0F	SI	$25	%	$3B	;	$51	Q	$67	g	$7D	}
$10	DLE	$26	&	$3C	<	$52	R	$68	h	$7E	~
$11	DC1	$27	’ (apost.)	$3D	=	$53	S	$69	i	$7F	DEL (delete)
$12	DC2	$28	(	$3E	>	$54	T	$6A	j		
$13	DC3	$29	)	$3F	?	$55	U	$6B	k		
$14	DC4	$2A	*	$40	@	$56	V	$6C	l		
$15	NAK	$2B	+	$41	A	$57	W	$6D	m		

Tabla 19. Conversión de Hexadecimal a ASCII

Conversión de Hexadecimal a Decimal

Para convertir un número hexadecimal (de hasta cuatro dígitos hexadecimales) a decimal, se puede mirar la equivalencia decimal de cada dígito hexadecimal en la Tabla 20. El decimal equivalente del número hexadecimal original es la suma de los pesos encontrados en la tabla, para todos los dígitos hexadecimales.

Por ejemplo: Para encontrar el número decimal equivalente de $3E7.

El decimal equivalente de 3 en el tercer dígito hex es 768.
El decimal equivalente de E en el segundo dígito hex es 224.
El decimal equivalente de 7 en el primer dígito hex es 7.

Luego sumando 768

$$\begin{array}{r} 224 \\ +\ \ 7 \\ \hline = 999 \end{array} \qquad \textbf{\$3E7 = } \mathbf{999_{10}}$$

15 bit 8				**7 bit 0**			
15 12		**11 8**		**7 4**		**3 0**	
4º dígito Hex		**3º dígito Hex**		**2º dígito Hex**		**1º dígito Hex**	
Hex	**Decimal**	**Hex**	**Decimal**	**Hex**	**Decimal**	**Hex**	**Decimal**
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,484	E	224	E	14
F	61,440	F	3,840	F	240	F	15

Tabla 20. Conversión de Hexadecimal a Decimal

Conversión de Decimal a Hexadecimal

Para convertir un número decimal (hasta $65{,}535_{10}$) a hexadecimal, se puede mirar el número decimal más grande, en la Tabla 20, que sea menor o igual al numero que se quiere convertir. El dígito hexadecimal correspondiente es el dígito hexadecimal más significativo del resultado. Se substrae al decimal el número encontrado del número decimal original, para conseguir el *valor decimal restante*. Se repite el procedimiento usando el valor decimal restante para cada dígito hexadecimal posterior.

Por ejemplo: Para encontrar el hexadecimal equivalente de 777_{10}

El número decimal más grande de la Tabla 20, que es menor o igual a 777_{10}, es 768_{10} Este corresponde a $3 en el tercer dígito hexadecimal.

Se substrae 768_{10} de 777_{10} para conseguir el valor decimal restante 9_{10}

Luego se mira en la columna el siguiente dígito hexadecimal de orden más bajo (en este caso el segundo dígito hex). Se encuentra el valor decimal más grande en esta columna, que sea menor o igual a 9_{10}, es 0, entonces se pondrá un 0 en el segundo dígito hex de su resultado.

9_{10} menos 0, es un valor decimal restante 9_{10}.

Luego se mira en la columna el siguiente dígito hexadecimal de orden más bajo (en este caso el primer dígito hex). Se encuentra el valor decimal más grande en esta columna, que sea menor o igual a 9_{10} es 9, por lo que se pondrá un 9 en el primer dígito hex de su resultado.

777_{10} = $309

Valores Hexadecimales vs. Instrucciones MC68HC05

La Tabla 21 muestra todos los valores hexadecimales, desde $00 a $FF y las instrucciones equivalentes del MC68HC05 con sus modos de direccionamiento. Hay sólo 210 instrucciones del MC68HC05, 46 de los valores hexadecimales no corresponden a una instrucción legal.

Operando	Instrucción	Modo de Direccionam	Operando	Instrucción	Modo de Direccionam	Operando	Instrucción	Modo de Dirección.
$00	BRSET0	Directo	$5A	DECX	Inherente	$AA	ORA	Inmediato
$01	BRCLR0	Directo	$5B	—	—	$AB	ADD	Inmediato
$02	BRSET1	Directo	$5C	INCX	Inherente	$AC	—	—
$03	BRCLR1	Directo	$5D	TSTX	Inherente	$AD	BSR	Relativo
$04	BRSET2	Directo	$5E	—	—	$AE	LDX	Inmediato
$05	BRCLR2	Directo	$5F	CLRX	Inherente	$AF	—	—
$06	BRSET3	Directo	$60	NEG	Indexado 1	$B0	SUB	Directo
$07	BRCLR3	Directo	$61	—	—	$B1	CMP	Directo
$08	BRSET4	Directo	$62	—	—	$B2	SBC	Directo
$09	BRCLR4	Directo	$63	COM	Indexado 1	$B3	CPX	Directo
$0A	BRSET5	Directo	$64	LSR	Indexado 1	$B4	AND	Directo
$0B	BRCLR5	Directo	$65	—	—	$B5	BIT	Directo
$0C	BRSET6	Directo	$66	ROR	Indexado 1	$B6	LDA	Directo
$0D	BRCLR6	Directo	$67	ASR	Indexado 1	$B7	STA	Directo
$0E	BRSET7	Directo	$68	LSL	Indexado 1	$B8	EOR	Directo
$0F	BRCLR7	Directo	$69	ROL	Indexado 1	$B9	ADC	Directo
$10	BSET0	Directo	$6A	DEC	Indexado 1	$BA	ORA	Directo
$11	BCLR0	Directo	$6B	—	—	$BB	ADD	Directo
$12	BSET1	Directo	$6C	INC	Indexado 1	$BC	JMP	Directo
$13	BCLR1	Directo	$6D	TST	Indexado 1	$BD	JSR	Directo
$14	BSET2	Directo	$6E	—	—	$BE	LDX	Directo
$15	BCLR2	Directo	$6F	CLR	Indexado 1	$BF	STX	Directo
$16	BSET3	Directo	$70	NEG	Indexado 0	$C0	SUB	Extendido
$17	BCLR3	Directo	$71	—	—	$C1	CMP	Extendido
$18	BSET4	Directo	$72	—	—	$C2	SBC	Extendido
$19	BCLR4	Directo	$73	COM	Indexado 0	$C3	CPX	Extendido
$1A	BSET5	Directo	$74	LSR	Indexado 0	$C4	AND	Extendido
$1B	BCLR5	Directo	$75	—	—	$C5	BIT	Extendido
$1C	BSET6	Directo	$76	ROR	Indexado 0	$C6	LDA	Extendido
$1D	BCLR6	Directo	$77	ASR	Indexado 0	$C7	STA	Extendido
$1E	BSET7	Directo	$78	LSL	Indexado 0	$C8	EOR	Extendido
$1F	BCLR7	Directo	$79	ROL	Indexado 0	$C9	ADC	Extendido
$20	BRA	Relativo	$7A	DEC	Indexado 0	$CA	ORA	Extendido
$21	BRN	Relativo	$7B	—	—	$CB	ADD	Extendido
$22	BHI	Relativo	$7C	INC	Indexado 0	$CC	JMP	Extendido
$23	BLS	Relativo	$7D	TST	Indexado 0	$CD	JSR	Extendido
$24	BCC	Relativo	$7E	—	—	$CE	LDX	Extendido

Operando	Instrucción	Modo de Direccionam	Operando	Instrucción	Modo de Direccionam	Operando	Instrucción	Modo de Direccionam
$25	BCS	Relativo	$7F	CLR	Indexado 0	$CF	STX	Extendido
$26	BNE	Relativo	$80	RTI	Inherente	$D0	SUB	Indexado 2
$27	BEQ	Relativo	$81	RTS	Inherente	$D1	CMP	Indexado 2
$28	BHCC	Relativo	$82	—	—	$D2	SBC	Indexado 2
$29	BHCS	Relativo	$83	SWI	Inherente	$D3	CPX	Indexado 2
$2A	BPL	Relativo	$84	—	—	$D4	AND	Indexado 2
$2B	BMI	Relativo	$85	—	—	$D5	BIT	Indexado 2
$2C	BMC	Relativo	$86	—	—	$D6	LDA	Indexado 2
$2D	BMS	Relativo	$87	—	—	$D7	STA	Indexado 2
$2E	BIL	Relativo	$88	—	—	$D8	EOR	Indexado 2
$2F	BIH	Relativo	$89	—	—	$D9	ADC	Indexado 2
$30	NEG	Directo	$8A	—	—	$DA	ORA	Indexado 2
$31	—	—	$8B	—	—	$DB	ADD	Indexado 2
$32	—	—	$8C	—	—	$DC	JMP	Indexado 2
$33	COM	Directo	$8D	—	—	$DD	JSR	Indexado 2
$34	LSR	Directo	$8E	STOP	Inherente	$DE	LDX	Indexado 2
$35	—	—	$8F	WAIT	Inherente	$DF	STX	Indexado 2
$36	ROR	Directo	$90	—	—	$E0	SUB	Indexado 1
$37	ASR	Directo	$91	—	—	$E1	CMP	Indexado 1
$38	LSL	Directo	$92	—	—	$E2	SBC	Indexado 1
$39	ROL	Directo	$93	—	—	$E3	CPX	Indexado 1
$3A	DEC	Directo	$94	—	—	$E4	AND	Indexado 1
$3B	—	—	$95	—	—	$E5	BIT	Indexado 1
$3C	INC	Directo	$96	—	—	$E6	LDA	Indexado 1
$3D	TST	Directo	$97	TAX	Inherente	$E7	STA	Indexado 1
$3E	—	—	$98	CLC	Inherente	$E8	EOR	Indexado 1
$3F	CLR	Directo	$99	SEC	Inherente	$E9	ADC	Indexado 1
$40	NEGA	inherente	$9A	CLI	Inherente	$EA	ORA	Indexado 1
$41	—	—	$9B	SEI	Inherente	$EB	ADD	Indexado 1
$42	—	—	$9C	RSP	Inherente	$EC	JMP	Indexado 1
$43	COMA	Inherente	$9D	NOP	Inherente	$ED	JSR	Indexado 1
$44	LSRA	Inherente	$9E	—	—	$EE	LDX	Indexado 1
$45	—	—	$9F	TXA	Inherente	$EF	STX	Indexado 1
$46	RORA	Inherente	$96	—	—	$F0	SUB	Indexado 0
$47	ASRA	Inherente	$97	TAX	Inherente	$F1	CMP	Indexado 0
$48	LSLA	Inherente	$98	CLC	Inherente	$F2	SBC	Indexado 0
$49	ROLA	Inherente	$99	SEC	Inherente	$F3	CPX	Indexado 0
$4A	DECA	Inherente	$9A	CLI	Inherente	$F4	AND	Indexado 0
$4B	—	—	$9B	SEI	Inherente	$F5	BIT	Indexado 0
$4C	INCA	Inherente	$9C	RSP	Inherente	$F6	LDA	Indexado 0
$4D	TSTA	Inherente	$9D	NOP	Inherente	$F7	STA	Indexado 0
$4E	—	—	$9E	—	—	$F8	EOR	Indexado 0
$4F	CLRA	Inherente	$9F	TXA	Inherente	$F9	ADC	Indexado 0
$50	NEGX	Inherente	$A0	SUB	Inmediato	$FA	ORA	Indexado 0
$51	—	—	$A1	CMP	Inmediato	$FB	ADD	Indexado 0
$52	—	—	$A2	SBC	Inmediato	$FC	JMP	Indexado 0
$53	COMX	Inherente	$A3	CPX	Inmediato	$FD	JSR	Indexado 0
$54	LSRX	Inherente	$A4	AND	Inmediato	$FE	LDX	Indexado 0
$55	—	—	$A5	BIT	Inmediato	$FF	STX	Indexado 0
$56	RORX	Inherente	$A6	LDA	Inmediato			
$57	ASRX	Inherente	$A7	—	—			
$58	LSLX	Inherente	$A8	EOR	Inmediato			
$59	ROLX	Inherente	$A9	ADC	Inmediato			

Tabla 21. Hexadecimales vs. Instrucciones Mnemónicas del MC68HC05

Glosario

1 K — Un kilobyte o 1024_{10} bytes. Similar al uso del prefijo kilogramo, que significa 1000 gramos en el sistema de numeración decimal. 1024 es 2^{10}.

8-bits MCU — Un microcontrolador donde los datos se comunican sobre un bus de datos compuesto de ocho líneas de datos separados. Todos los miembros de la familia de microcontroladores MC68HC05 son MCUs de 8-bits.

A — Es la abreviación de *Acumulador* en las MCU de los MC68HC05.

accumulator (acumulador) — Es un registro de 8-bits de la CPU del MC68HC05. Los contenidos de este registro se pueden usar como un operando de una instrucción aritmética o nivel lógico.

addressing mode (modo de direccionamiento) — Es la manera en que la CPU obtiene (direcciones) la información necesaria para completar una instrucción. Las CPU de la familia MC68HC05 tienen seis modos de direccionamiento:

- **Inerente** — La CPU no necesita información adicional de la memoria para completar la instrucción.
- **Inmediato** — La información necesaria para completar la instrucción se localiza en las siguientes posiciones de memoria, después del 'opcode'.
- **Directo** — El byte de orden más bajo de la dirección del operando, se localiza en la siguiente posición de memoria después del 'opcode' y el byte de orden más alto de la dirección del operando, se asume que es $00.
- **Extendido** — El byte de orden más alto de la dirección del operando se localiza en la siguiente posición de memoria después del 'opcode' y el byte de orden más bajo de la dirección del operando se localiza en la siguiente posición de memoria después de este operando.
- **Indexado** — La dirección del operando depende del valor actual en el registro de índice X y el valor que proporciona una instrucción de 0-, 8- o 16-bits.
- **Relativo** — Usado para las instrucciones de bifurcación para especificar la dirección de destino, donde el proceso continuará si la condición de bifurcación es verdad.

address bus (bus de direcciones) — El conjunto de líneas que son usadas para seleccionar una posición de memoria específica, para que la CPU pueda escribir información en la posición de memoria o leer sus contenidos. Si un microcontrolador tiene 11 líneas en su bus de direcciones, se pueden direccionar 2^{11} o 2048_{10} posiciones de memoria. En la mayoría de MCUs de la familia MC68HC05, el bus de direcciones no es accesible a los pines externos.

ALU — Son las siglas de *Unidad Aritmético Lógica*. Es la porción de la CPU de un microprocesador donde tienen lugar las operaciones matemáticas y lógicas. Otra circuitería descodifica cada instrucción y configura la ALU para realizar las necesarias operaciones aritméticas o lógicas en cada paso de una instrucción.

ASCII — Son las siglas de American Standard Code for Information Interchange (código estándar americana para el intercambio de información). Una correlación aceptada ampliamente entre los caracteres alfabéticos y numéricos, especificados con números binarios de 7-bits. Referirse a la **Tabla 19. Conversión de ASCII a Hexadecimal**.

analog (analógica) — Es una señal que puede tener valores de nivel de voltaje que ni no son el nivel V_{SS} ni el nivel V_{DD}. Para que un microprocesador pueda usar esas señales, deben convertirse en un número binario al que corresponde el nivel de voltaje de la señal. Se puede usar un convertidor analógico a digital para realizar esta conversión. Por contra, una señal digital sólo tiene dos posibles valores, 1 ($\approx V_{DD}$) o 0 ($\approx V_{SS}$).

application programs (programas de aplicación) — Es el software de aplicación que instruye a un microprocesador para que resuelva un problema de aplicación.

arithmetic logic unit (unidad aritmética lógica) — Es la porción de la CPU de un microprocesador donde tienen lugar las operaciones matemáticas y lógicas. Otra circuitería descifra cada instrucción y configura la ALU para realizar las necesarias operaciones aritméticas o lógicas en cada paso de una instrucción.

assembler (ensamblador) — Es un programa de software que traduce el código fuente mnemónico en 'opcodes', que después se podrá cargar en la memoria de un microcontrolador.

assembly language (lenguaje ensamblador) — Son las instrucciones mnemónicas y las directivas del ensamblador que son significativas para programar y se pueden traducir en un programa de código objeto que un microcontrolador entiende. La CPU usa los 'opcodes' y los números binarios para especificar las operaciones que constituyen un programa de microprocesador. Éstos números no son significativos a las personas, por esto los programadores usan el lenguaje ensamblador para representar las instrucciones. Las directivas del ensamblador proporcionan información adicional como la posición de memoria para iniciar un programa. Se usan etiquetas para significar una dirección o un valor binario.

base 2 — Son números binarios que usan sólo dos dígitos, 0 y 1. La base 2 es el sistema de numeración que usan las computadoras.

base 10 — Son números decimales que usan 10 dígitos, desde el 0 hasta el 9. Este es el sistema de numeración usado por las personas.

base 16 — Es el sistema de numeración hexadecimal. Los 16 caracteres (del 0 al 9 y las letras de A a F) se usan para representar el valor hexadecimal. Un dígito hexadecimal puede representar exactamente un valor binario de 4-bits. El sistema de numeración hexadecimal es usado por las personas para representar valores binarios, porque es más fácil de usar un número de 2 dígitos que el número binario equivalente de 8 dígitos. Referirse a la **Tabla 1. Equivalencias entre el Decimal, Binario y Hexadecimal**.

BCD — El *Binario Codificado en Decimal* es una anotación que usa valores binarios para representar cantidades decimales. Cada dígito BCD usa cuatro bits binarios. Se consideran ilegales seis de las 16 combinaciones binarias posibles.

binario — Es el sistema de numeración usado por las computadoras, cualquier cantidad puede ser representada por una serie de 1 y 0. Eléctricamente, estos 1 y 0 están representados por niveles de voltaje VDD y VSS respectivamente.

bit — Es un simple dígito binario. Un bit sólo puede tener un sólo valor, 0 o 1.

black box (caja negra) — Es un hipotético bloque de nivel lógico o de circuitería que realiza alguna transformación de entrada a salida. Una caja negra se usa cuando la relación de entrada a salida es conocida, pero los medios para lograr esta transformación no es conocida o no es importante de discutir.

branch instructions (instrucciones de bifurcación) — Son instrucciones que causan a la CPU que continúe procesando hacia una posición de memoria que la siguiente dirección secuencial. La mayoría de instrucciones de bifurcación son condicionales. Es decir, la CPU continuará a la siguiente dirección secuencial (de no bifurcación) si una condición es falsa o continúa a otra dirección (de bifurcación) si la condición es verdad.

breakpoint (punto de paro) — Durante un programa de depuración, es útil correr las instrucciones hasta conseguir que la CPU se pare en un lugar específico del programa y entonces entrar en un programa de depuración. Se establece un punto de paro en la dirección deseada, sustituyendo temporalmente una interrupción por software (SWI) para la instrucción en esa dirección. En respuesta a la SWI, el control se pasa al programa de depuración.

byte — Es un conjunto de ocho bits binarios.

C — Es la abreviación para *Carry/borrow* en el registro de código de condición del MC68HC05. Al agregar dos números de 8-bits sin signo, el bit C se pone a 1, si el resultado es mayor que 255 ($FF).

CCR — Son las siglas de R*egistro de Código de Condición* en el MC68HC05. El CCR tiene cinco bits (H, I, N, Z, y C) que se pueden usar para controlar las instrucciones de bifurcación condicionales. Los valores de los bits en el CCR están determinados por los resultados de las operaciones anteriores. Por ejemplo, después de una instrucción de carga del acumulador (LDA), Z se pondrá a 1, si el valor cargado fue $00.

central processor unit (unidad central de proceso) — Es la parte central de una computadora que controla la ejecución de las instrucciones.

checksum (suma de prueba) — Es un valor, que es el resultado de sumar una serie de números binarios. Al intercambiar información entre computadoras, el 'checksum' da una indicación sobre la integridad de los datos

transferidos. Si se transfieren los valores incorrectamente, es muy improbable que el 'checksum' empareje el valor que se espera.

clock (reloj) — Es una señal cuadrada que se usa para secuenciar los eventos de una computadora.

CMOS — Son las siglas de *Semiconductor de Óxido de Metal Complementario*. Es un semiconductor de silicio procesado tecnológicamente que permite la fabricación de dos tipos de transistores, el tipo N y el tipo P, en el mismo circuito integrado. La mayoría de microcontroladores modernos usan la tecnología CMOS.

computer program (programa) — Es serie de instrucciones que causan a una computadora que haga algo.

computer system (sistema microprocesador) — Es una CPU, más otros componentes que necesitan realizar una función útil. Un sistema microprocesador mínimo incluye una CPU, el reloj, la memoria, un programa y interfaces de entrada/salida.

condition code register (registro de código de condición) — El registro de código de condición CCR tiene cinco Bits (H, I, N, Z, y C) que pueden ser usados para controlar instrucciones de bifurcación condicional. Los valores de los bits en el CCR son determinados por los resultados de las operaciones anteriores. Por ejemplo, después de una instrucción de carga del acumulador (LDA), Z se pondrá a 1, si el valor cargado fue $00.

CPU — Son las siglas de *Unidad Central de Proceso*. Parte central de una computadora que controla la ejecución de las instrucciones.

CPU cycles (ciclos de la CPU) — Un ciclo de reloj de la CPU es uno periodo de reloj del bus interno. Normalmente, este reloj es derivado dividiendo la frecuencia generada por un oscilador, por dos o más, con los tiempos altos y bajos iguales. La longitud de tiempo requerido para ejecutar una instrucción es medida en ciclos de reloj de la CPU.

CPU registers (registros de la CPU) — Son las posiciones de memoria que se conectan directamente a un nivel lógico de la CPU, en lugar de ser parte direccionable del mapa de memoria. La CPU siempre tiene acceso directo a la información de éstos registros. Los registros de la CPU en el MC68HC05 son:

- **A** — acumulador de 8-bits.
- **X** — registro de índice de 8-bits.
- **CCR** — registro de código de condición, que contiene los bits H, I, N, Z y C.
- **SP** — puntero de pila.
- **PC** — contador de programa.

CRT — Son las siglas de *Tubo de Rayos Catódicos*. También usada como una expresión informal para referirse a un terminal de comunicación completo que tiene un teclado y una pantalla de vídeo.

cycles (ciclos) — Ver ciclos de la CPU.

data bus (bus de datos) — Es un conjunto de líneas que se usan para llevar la información binaria desde la CPU a una posición de memoria o de una posición de memoria a la CPU. En el MC68HC05, el bus de datos es de ocho bits.

decimal — Son números de base 10, usa los dígitos del 0 al 9. Este sistema de numeración es normalmente usado por los humanos.

development tools (herramientas de desarrollo) — Son dispositivos de software o hardware, que se usan para desarrollar el programa y el hardware de la aplicación. Como ejemplos de herramientas de desarrollo de software son: los editores del texto, ensambladores, monitores de depuración y simuladores. Como ejemplos de herramientas de desarrollo de hardware son: los emuladores, analizadores lógicos y programadores PROM. Un simulador en circuito combina un simulador software con interfaces hardware.

digital — Es un sistema lógico binario donde las señales sólo pueden tener dos estados, 0 ($\approx V_{SS}$) o 1 ($\approx V_{DD}$).

dircet address (dirección directa) — Es una dirección dentro de las primeras 256 direcciones de memoria ($0000 a $00FF). El byte de orden alto de estas direcciones siempre es $00. Las instrucciones especiales permiten que estas direcciones sean accedidas usando sólo el byte de orden bajo de su dirección. Éstas instrucciones automáticamente se fijan con el valor asumido $00 para el byte de orden alto de la dirección.

direct addressing mode (modo de direccionamiento directo) — El modo de direccionamiento directo usa un valor proporcionado por el programa para el byte de orden bajo de la dirección de un operando. El byte de orden alto de la dirección del operando se asume como $00, que no tiene que ser especificado explícitamente.

direct page (página directa) — Los primeros 256 bytes de memoria, $0000 a $00FF.

EEPROM —Memoria programable de sólo lectura y borrable eléctricamente. Un tipo de memoria no volátil que puede borrarse y reprogramarse por instrucciones de programa. No necesita ningún tipo de tensión especial o luz ultra violeta, los contenidos de este tipo de memoria se pueden cambiar sin quitarla del sistema de aplicación.

effective address (dirección eficaz) — Es la dirección donde se localiza un operando de la instrucción. El modo de direccionamiento de una instrucción, determina cómo la CPU calcula la dirección eficaz del operando.

Embedded — Cuando un circuito de aplicación contiene un microcontrolador, la MCU se dice que es un controlador 'embedded' (incluido). A menudo, el usuario final del aparato no es consciente que hay una computadora dentro.

EPROM — Es una memoria programable sólo de lectura y borrable por luz ultravioleta. Se reconocen fácilmente las MCU que tienen EPROM porque el encapsulado tiene una ventana de cuarzo para permitir la exposición a la luz ultravioleta. Si una MCU EPROM se encapsula en material de plástico opaco, se llama OTP (One Time Programmable) porque no hay ninguna manera de exponer la EPROM a luz ultravioleta.

extended addressing mode (modo de direccionamiento extendido) — En el modo de direccionamiento extendido, el byte de orden alto de la dirección del operando se localiza en la siguiente posición memoria después del 'opcode'. El byte de orden bajo de la dirección del operando se localiza en la segunda posición de memoria después el 'opcode'.

fetching a vector (sacando un vector) — Cuando se hace un 'reset' a la CPU o responde a una interrupción, se cargan en el contador de programa los contenidos de un par de posiciones de memoria específicas y continúa procesando desde la dirección cargada. El proceso de lectura de estas dos posiciones, se llama 'sacando el vector'.

flowchart (diagrama de flujo) — Es un diagrama que muestra la sucesión de pasos requeridos para realizar una operación. Un diagrama de flujo no sólo dice *que* se necesita hacer, pero también el *orden* en que los pasos se deben hacer.

H — Es la abreviación de medio acarreo (*H*alf carry) en el registro de código de condición del MC68HC05. Este bit indica un acarreo desde los cuatro bits de orden bajo de un valor de 8-bits, a los cuatro bits de orden alto. Este indicador de estado se usa durante los cálculos BCD.

half flip-flop (medio flip-flop) — El medio flip-flop (HFF) tiene una condición de *transparente* y una condición de '*latch*'. En la condición de transparente (entrada de reloj igual a un nivel lógico 1), la salida de Q es siempre a un nivel lógico igual al presentado en la entrada. En la condición de 'latch' (la entrada de reloj es igual a un nivel lógico 0), la salida mantiene el nivel lógico que estaba presente cuando el flip-flop estaba en la última en la condición de transparente.

hexadecimal — Es el sistema de numeración hexadecimal. Los 16 caracteres (del 0 al 9 y las letras de A a F) se usan para representar el valor hexadecimal. Un dígito hexadecimal puede representar exactamente un valor binario de 4-bits. El sistema de numeración hexadecimal es usado por las personas para representar valores binarios, porque es más fácil de usar, un número de 2 dígitos, que el número binario equivalente de 8 dígitos. Referirse a la **Tabla 1. Equivalencias entre el Decimal, Binario y Hexadecimal**.

high order (orden alto) — Es el dígito más a la izquierda de un número. El dígito cinco es el orden alto del número 57.

I — Abreviación *bit de máscara de Interrupción*, en el registro de código de condición del MC68HC05.

I/O (E/S) — Input/output (Entrada/Salida). Las E/S interconectan un sistema microprocesador y el mundo externo. La CPU lee una entrada para detectar el nivel de señal externa y escribe a una salida para cambiar el nivel de una señal externa.

immediate addressing mode (modo de direccionamiento inmediato) — En el modo de direccionamiento inmediato, el operando se localiza en las próximas posiciones de memoria, después del 'opcode'.

inherent addressing mode (modo de direccionamiento inherente) — En el modo de direccionamiento inherente, la CPU ya inherentemente sabe todo lo que necesita saber para completar la instrucción. Los operandos (si hay alguno) están en los registros de la CPU.

in circuit simulator (simulador en circuito) — Es un simulador con interfaces de hardware que permite la conexión en un circuito de aplicación. El simulador en circuito reemplaza la MCU y se comporta tal como lo haría una MCU real. El diseñador tiene mayor control y visibilidad de las operaciones internas de la MCU, porque están siendo simuladas por instrucciones en el ordenador personal del diseñador. Un simulador en circuito, como otros simuladores, no es tan rápido como una MCU real.

indexed addressing mode (modo de direccionamiento indexado) — En el modo de direccionamiento indexado, el valor actual del registro de índice se agrega a un valor de 0-, 8- o 16-bits en la instrucción, para conseguir la dirección eficaz del operando. Hay 'opcodes' separados para variaciones de 0-, 8-, y 16-bits, de instrucciones de modo indexado, para que la CPU sepa cuántas posiciones de memoria adicionales, para leer después del 'opcode'.

index register X (registro de índice X) — Es un registro de la CPU de 8-bits que se usa en el modo de direccionamiento indexado. X es la abreviación, también puede usarse como un registro de propósito general de 8-bits (además del acumulador).

input/output (entrada/salida) — Es la interface entre el microcontrolador y el mundo externo. La CPU lee una entrada para detectar el nivel de señal externa y escribe a una salida para cambiar el nivel de una señal externa.

instruction decoder (decodificador de la instrucción) — Es la parte de la CPU que recibe un 'opcode' y produce las señales de control necesarias para que el resto de la CPU realice las operaciones deseadas.

instructions (instrucciones) — Las instrucciones son operaciones que la CPU puede realizar. Las instrucciones son expresadas por programas como lenguaje ensamblador mnemónico. La CPU interpreta un 'opcode' y sus operandos asociados como una instrucción.

instruction set (juego de instrucciones) — El conjunto de instrucciones de una CPU son el conjunto de todas las operaciones que la CPU sabe realizar. Una manera para representar el conjunto de instrucciones, es con los mnemónicos, como LDA, que significa cargar el acumulador A. Otra representación del conjunto de instrucciones, es con los 'opcodes' que son las instrucciones que reconoce la CPU.

inverter (inversor) — Es un circuito lógico simple que produce un nivel de salida que es el opuesto del nivel presentado a su entrada.

kilobyte — 1 kilobyte es 1024_{10} bytes. Es similar al uso del prefijo kilogramo que significa 1000 gramos, en el sistema de numeración decimal. 1024 es 2^{10}.

label (etiqueta) — Es un nombre que se asigna (por un programador) a una dirección específica o al valor binario. Cuando un programa contiene una etiqueta y es ensamblado, la etiqueta es reemplazada por el valor binario que representa. Los programas típicamente incluyen muchas etiquetas.

latch — Es un circuito lógico que mantiene una salida estable después de que la entrada ha estado quitada o cambiada. Una entrada de reloj de control, determina cuando el 'latch' capturará el estado de entrada y lo aplicará a la salida.

least significant bit (bit menos significativo) — LSB, es el dígito más a la derecha de un valor binario.

listing (listado) — Un listado de un programa, muestra los números binarios que la CPU necesita junto a las declaraciones en lenguaje ensamblador, que escribió el programador. El listado es generado por el ensamblador en el proceso de traducción del ensamblado de las declaraciones del lenguaje fuente en información binaria que necesita la CPU.

logic 1 (estado lógico 1) — Es un nivel de voltaje aproximadamente igual a V_{DD} (positivo).

logic 0 (estado lógico 0) — Es un nivel de voltaje aproximadamente igual a V_{SS} (masa).

low order (orden bajo) — Es el dígito más a la derecha de un número. Siete es el dígito de orden bajo del número 57.

LSB — Son las siglas de Least Significant Bit. Es el bit menos significativo. El dígito de más a la derecha de un valor binario.

machine codes (código máquina) — Son los códigos binarios que son procesados por la CPU como instrucciones. El código máquina incluye los 'opcodes' y los operandos.

mainframe computer — Es un sistema ordenador grande que normalmente está situado en un cuarto especial. Las computadoras 'mainframe' se usan para procesar grandes cantidades de información o trabajos de mantenimiento de bases de datos, por ejemplo, bases de datos de todos los asegurados de una compañía de seguros.

mass storage (almacenamiento masivo) — Es un dispositivo de almacenamiento de muy alta capacidad, por ejemplo, un disco magnético. La información en un dispositivo de almacenamiento masivo, toma mucho más tiempo para acceder a la información, que en el mapa de memoria de la CPU.

MCU — Son las siglas de *Unidad Micro Controlador*. Es un sistema microprocesador completo, con la CPU, la memoria, el oscilador de reloj y las E/S, en un solo circuito integró.

memory location (posición de memoria) — En la memoria del MC68HC05, cada posición de memoria guarda un byte de datos y tiene una única dirección. Para guardar la información en una posición de memoria, la CPU pone la dirección de la posición en el bus de direcciones, la información de datos en el bus de datos y se afirma con una señal 'write' (escritura). Para leer la información de una posición de memoria, la CPU pone la dirección de la posición en el bus de direcciones y se afirma con la señal 'read' (lectura). En respuesta a la señal de lectura, la posición de memoria seleccionada pone sus datos hacia el bus de datos.

memory map (mapa de memoria) — Es una representación pictórica de todas las posiciones de memoria en un sistema microcontrolador. Un mapa de memoria es similar en una ciudad el mapa de calles, en que se muestra donde se localizan las cosas.

memory mapped I/O (trazado del mapa de memoria de E/S) — En este tipo de sistema las E/S y los registros de control se acceden de la misma manera, como las posiciones de memoria RAM o ROM. Cualquier instrucción que puede usarse para acceder a la memoria, también se puede usar para acceder a los registros de E/S.

microcontroller (microcontrolador) — Es un sistema microprocesador completo, incluyendo la CPU, la memoria, el oscilador de reloj y las E/S, en un sólo circuito integrado.

microprocessor (microprocesador) — Un microprocesador es similar a un microcontrolador, sólo que uno o más de los subsistemas necesarios para hacer un sistema completo, no están incluidos en el mismo circuito integrado de la CPU. Un microprocesador incluye típicamente la CPU y un reloj oscilador, pero no incluye la memoria de programa o los registros de E/S.

mnemonic (mnemónico) — Tres de las cinco letras que representan una operación en un microprocesador. Por ejemplo, la forma mnemónica de carga del acumulador, es la instrucción LDA.

monitor program (programa monitor) — Es un programa de software que está pensado para ayudar al desarrollo del sistema. Un programa monitor típico permite a un usuario examinar y cambiar los contenidos de la memoria o de los registros de la CPU, poner puntos de paro (breakpoints) y selectivamente ejecutar programas de aplicación.

most significant bit (bit más significativo) — MSB es el dígito de más a la izquierda de un valor binario.

MSB — Son las siglas de Most Significant Bit (el bit más significativo). El dígito de más a la izquierda de un valor binario.

N — Es la abreviación de *Negativo*, es un bit en el registro de código de condición del MC68HC05. En anotación de complemento a dos, los números positivos con signo tienen un 0 en su MSB y los números negativos tienen un 1 en su MSB. El bit N del registro de código de condición refleja el signo del resultado de

una operación. Después de una instrucción de carga del acumulador, el bit N se pondrá a 1 si el MSB del valor cargado fue un 1.

NAND gate (puerta NAND) — Es un circuito lógico básico. La salida de una puerta NAND está a un nivel lógico 0 cuando todas sus entradas están a un nivel lógico 1. La salida de una puerta NAND está a un nivel lógico 1 si cualquiera de sus entradas están a un nivel lógico 0.

non volatile (no volatil) — Es un tipo de memoria que no se olvida de sus contenidos cuando se desconecta la alimentación. Las ROM, EPROM y EEPROM son memorias no volátiles.

NOR gate (puerta NOR) — Es un circuito lógico básico. La salida de una puerta NOR está a un nivel lógico 0 cuando cualquiera de sus entradas están a un nivel lógico 1. La salida de una puerta NOR está a un nivel lógico 1 si todas sus entradas están a un nivel lógico 0.

object code file (archivo de código objeto) — Es un archivo de texto que contiene números que representan los 'opcodes' binarios y datos de un programa para un microprocesador. Un archivo de código de objeto se puede usar para cargar la información binaria en un sistema microcontrolador. Motorola usa el formato de archivo S-record para los archivos de código objeto. Ver la Figura 35. Archivo S-record del programa ejemplo.

octal — Son los números de base 8, que usan los caracteres del 0 al 7, representan conjuntos de tres bits binarios. El sistema octal raramente se usa.

one (uno) — Es un nivel lógico alto ($\approx V_{DD}$).

ones complement (unos complementarios) — Para conseguir el uno complementario lógico de un valor binario, hay que invertir cada bit.

opcode (código de operación) — Es un código binario que le dice a la CPU que haga una operación específica de una manera específica. La CPU del MC68HC05 reconoce 210 únicos 'opcodes' de 8-bits que representan variaciones de modo de direccionamiento de 62 instrucciones básicas.

operand (operando) — Es un valor de entrada en una operación lógica o matemática.

oscillator (oscilador) — Es un circuito que produce una frecuencia constante de onda cuadrada, que se usa en el microprocesador como cronometro y como secuenciador de referencia. Un microcontrolador incluye todos los elementos de esto circuito, excepto los componentes que determinan la frecuencia (el cristal o el R-C, resistencia-condensador).

OTP Son las siglas de *One Time Programmable* (programable una sola vez) — Ver OTPROM.

OTPROM — Es un tipo de memoria no volátil que puede programarse, pero no se puede borrar. Una OTPROM es un MCU EPROM que se ha encapsulado en plástico opaco. Se llama una MCU programable una sola vez, porque no hay ninguna manera de exponer la EPROM a luz ultravioleta.

parity (paridad) — Es un bit extra, en una palabra binaria, que está pensada para indicar la validez de los bits restantes de la palabra. En paridad par, el bit de paridad se pone a 1 o 0 como una necesidad de hacer el número total de nivel lógico 1 en la palabra (incluyendo el bit de paridad) igual a un número par (0, 2, 4, etc.).

PC — Son las siglas de *Program Counter* (contador de programa), un registro de la CPU de la MCU del MC68HC05. También se usa siglas de Personal Computer (ordenador personal).

personal computer (ordenador personal) — Un sistema de computadora pequeña que normalmente se usa para el uso de una persona para procesar información.

playing computer — Una técnica de aprendizaje en la que el propio usuario pretende ser una CPU que está ejecutando las instrucciones de un programa.

pointer register (registro puntero) — Es un registro de índice que a veces se llama registro puntero, porque sus contenidos se usan para el cálculo de la dirección de un operando. Por ejemplo, es poner en una instrucción indexada sin desplazamiento, donde el registro X contiene la dirección directa del operando.

program (programa) — Es un conjunto de instrucciones que causan que un ordenador realice una tarea de aplicación.

program counter (contador de programa) — El contador de programa (PC) es el registro de la CPU que almacena la dirección de la siguiente instrucción o el operando que usa la CPU.

programming model (modelo de programación) — Son los registros de una CPU en particular. El modelo de programación de la CPU de la familia MC68HC05 se muestra en la **Figura 23. Modelo de programación**.

PROM — Es una memoria programable de sólo lectura. Un tipo de memoria no volátil que puede programarse después de ser fabricada. EPROM y EEPROM son dos tipos de memorias PROM.

pulled (sacado) — Es el acto de leer un valor en la pila. En el MC68HC05, un valor es sacado por esta secuencia de operaciones: Primero, el registro del puntero de pila se incrementa para que apunte al último valor que se ha guardado en la pila. Después, el valor que está en la dirección contenida en el registro del puntero de pila, es leído por la CPU.

pushed (guardado) — Es el acto de guardar un valor a la dirección contenida en el registro del puntero de pila y después se decrementa el puntero de pila para que apunte a la siguiente posición disponible de la pila.

RAM — Es una memoria de acceso aleatorio. Cualquier posición de la RAM se puede leer o escribir por la CPU. El contenido de una posición de memoria de la RAM permanecerá válida, hasta que la CPU escriba un valor diferente o hasta desconectar la alimentación.

read (lectura) — Transfiere los contenidos de una posición de memoria a la CPU.

record (registro) — Es una línea de un archivo de texto de código objeto. Ver S-record.

registers (registros) — Son las posiciones de memoria que se conectan directamente en el nivel lógico de la CPU en lugar de ser una parte direccionable del mapa de memoria. La CPU siempre tiene acceso directo a la información en estos registros. Los registros de la CPU MC68HC05 son:

- A — acumulador de 8 bits
- X — registro de índice de 8 bits
- CCR — registro de código de condición que contiene los bits H, I, N, Z y C
- SP — puntero de pila
- PC — contador de programa

Las posiciones de memoria que almacenan el estado y la información de control para periféricos internos del circuito integrado son llamadas E/S y registros de control.

relative addressing mode (modo de direccionamiento relativo) — El modo de direccionamiento relativo se usa para calcular la dirección de destino para las instrucciones de bifurcación. Si la condición de bifurcación es verdad, el valor con signo de los 8-bits después del 'opcode' es agregado al valor actual del contador de programa para conseguir la dirección donde la CPU sacará la siguiente instrucción.

relative offset (desplazamiento relativo) — Unos 8-bits, un valor complemento a dos con signo que se agrega al contador de programa cuando una condición de bifurcación es verdad. El desplazamiento relativo se localiza en el byte después de un 'opcode' de bifurcación.

reset — El 'reset' se usa para forzar un sistema microprocesador que arranque en un punto conocido y para forzar a los periféricos internos, para arrancar en condiciones conocidas.

reset vector (vector de reset) — Los contenidos de las últimas dos posiciones de memoria en una MCU del MC68HC05, se les llama vector de 'reset'. Así que la MCU deja el 'reset', el contador de programa se carga con el contenido de éstas dos posiciones de memoria, para que la primera instrucción después del 'reset' sea sacada desde esta dirección.

ROM — Es una memoria de sólo lectura. Un tipo de memoria que puede leerse pero no puede modificarse (escribirse). Los contenidos de la ROM deben ser especificados antes de fabricar la MCU.

S-record — Un formato estándar de Motorola que se usa para los archivos de código objeto. Ver la *Figura 35. Archivo S-record del Programa Ejemplo*.

simulator (simulador) — Es un programa que simula la conducta real de una MCU.

source code (código fuente) — Ver source program (programa fuente).

source program (programa fuente) — Es un archivo texto que contiene los mnemónicos de cada instrucción, las etiquetas, los comentarios y las directivas del ensamblador. El archivo fuente es procesado por un ensamblador para producir una listado compuesto y una representación de un archivo objeto del programa.

SP — Siglas de 'stack pointer' (puntero de pila), un registro de la CPU en la MCU del MC68HC05.

stack (pila) — Es un mecanismo para guardar temporalmente los valores de los registros de la CPU durante las interrupciones y subrutinas. La CPU mantiene esta estructura con el registro del puntero de pila que contiene la dirección de la siguiente posición del almacenamiento disponible en la pila. Cuando se hace una llamada a una subrutina, la CPU guarda el orden bajo y los bytes de orden alto de la dirección de retorno en la pila, antes de empezar las instrucciones de la subrutina. Cuando la subrutina termina, una instrucción de retorno de subrutina (RTS) causa que la CPU recupere la dirección de retorno de la pila y que continúe procesando desde donde salió antes de la subrutina. Las interrupciones trabajan de la misma manera excepto todos los registros de la CPU, que se guardan en la pila en lugar del contador de programa.

stack pointer (puntero de pila) — Es un registro de la CPU que almacena la dirección de la siguiente posición del almacenamiento disponible en la pila.

subrutine (subrutina) — En el curso de un programa se necesitan ser usadas unas sucesiones de instrucciones. La última instrucción en la subrutina es una instrucción de retorno de subrutina (RTS). En un lugar del programa principal, donde son necesitadas las instrucciones de subrutina, se usa una instrucción de salto o bifurcación a subrutina (JSR o BSR) para llamar a la subrutina. La CPU deja el flujo del programa principal para ejecutar las instrucciones en la subrutina. Cuando la instrucción RTS se ejecuta, la CPU vuelve al programa principal de donde había salido.

three state buffer ('buffer' de tres estados) — La salida del 'buffer' de tres estados, puede ser un estado lógico 0, un estado lógico 1 o un estado de alta impedancia (como si no hubiese nada conectado). Una entrada de control habilita el estado de alta impedancia (off) vs. el estado de baja impedancia (on). Cuando el 'buffer' está habilitado, la salida tiene el mismo nivel que el de la entrada (1 o 0). Cuando el 'buffer' no está habilitado, la salida actúa como un circuito abierto.

transducer (transductor) — Es un dispositivo que convierte alguna propiedad física, como presión, en señales eléctricas que pueden ser usadas por una computadora.

transmission gate (puerta de transmisión) — Es un circuito lógico básico usado en microcontroladores. Una puerta de transmisión trabaja como un interruptor en serie, está controlado por una señal de nivel lógico. Cuando la entrada de control está a un nivel lógico 0, la puerta de transmisión actúa como un circuito abierto. Cuando la entrada de control es un nivel lógico 1, la puerta de transmisión actúa como un corto circuito.

twos complement (complemento a dos) — Es un medio para realizar una substracción binaria usando técnicas de suma. El bit más significativo de un número complemento a dos, indica el signo del número (1 indica negativo). El negativo de un número complemento a dos, se obtiene invirtiendo cada bit del número y después añadiendo 1 al resultado. Por ejemplo, el negativo del complemento a dos de 0000 0011 (3_{10}) es 1111 1100 + 0000 0001 = 1111 1101.

variable — Es un valor que cambia durante el curso de la ejecución de un programa.

V_{DD} — Es la alimentación positiva de un microcontrolador, típicamente es de 5 Vdc.

vector — Es un indicador (de dirección) que indica donde la CPU debe continuar procesando las instrucciones después de una interrupción o de un 'reset'.

V_{SS} — Es la alimentación de 0 Vdc de un microcontrolador.

volatile (volátil) — Es un tipo de memoria que pierde los contenidos cuando desaparece la alimentación. La RAM es un tipo de memoria volátil. Los microcontroladores modernos, necesitan muy poca corriente para mantener los contenidos de una RAM bajo las condiciones normales de funcionamiento. En algunos casos, los

contenidos de la RAM y los registros pueden permanecer inalterados después de un pequeño corte de alimentación.

word (palabra) — Es un grupo de bits binarios. Algunas computadoras más grandes consideran una palabra un conjunto de 16 bits, pero ésta no es una norma universal.

write (escritura) — Es la transferencia de un byte de datos de la CPU a una posición de memoria.

X — Abreviación de *indeX register* (registro de índice), un registro de la CPU en la MCU del MC68HC05.

Z —Abreviación de *Zero* (cero), un bit del registro de código de condición del MC68HC05. Una instrucción de comparación substrae los contenidos del valor probado de un registro. Si los valores fueran iguales, el resultado de esta substracción sería cero, por lo que el bit Z se pondría a 1. Después de una instrucción de carga del acumulador, el bit Z se pondrá a 1 si el valor cargado era $00.

zero (cero) — Es un nivel lógico bajo (V_{SS}).

zero crossings (cruce por cero) — Cuando una señal de corriente alterna va de un valor positivo a negativo o de negativo a positivo, en ese punto se llama un cruce por cero. Los 50-Hz de un línea de corriente alterna, cruza por cero cada 10 milisegundos.