Microprocesadores y microcontroladores 3

Anexos

MINISTERIO de EDUCACIÓN CIENCIA y TECNOLOGÍA PRESIDENCIA de la NACIÓN

inet Instituto Nacional de Educación Tecnológica

serie/desarrollo de contenidos
colección/fluídica y controladores lógicos programables

Autoridades

Presidente de la Nación
Néstor C. Kirchner

Ministro de Educación, Ciencia y Tecnología
Daniel Filmus

Directora Ejecutiva del Instituto Nacional de Educación Tecnológica
María Rosa Almandoz

Director Nacional del Centro Nacional de Educación Tecnológica
Juan Manuel Kirschenbaum

Especialista en contenidos

- Marcelo Estévez

Ministerio de Educación, Ciencia y Tecnología
Instituto Nacional de Educación Tecnológica
Saavedra 789. C1229ACE
Ciudad Autónoma de Buenos Aires
República Argentina

serie/desarrollo de contenidos

Colecciones

- Autotrónica
- Comunicación de señales y datos
- Diseño gráfico industrial
- Electrónica y sistemas de control
- Fluídica y controladores lógicos programables
    1. Tecnología neumática
    2. Controladores lógicos programables –PLC–
    3. Microprocesadores y microcontroladores
- Gestión de la calidad
- Gestión de las organizaciones
- Informática
- Invernadero computarizado
- Laboratorio interactivo de idiomas
- Procesos de producción integrada
- Proyecto tecnológico
- Unidades de cultura tecnológica

Índice

ANEXOS

Sistemas de numeración

Un sistema de numeración queda unívocamente definido a través de su base, la cual indica la cantidad de dígitos distintos que posee el sistema, que se pueden utilizar para escribir cualquier número dentro de ese sistema.

Ejemplos de sistemas de numeración son:

- Decimal, base 10 (dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
- Binario, base 2 (dígitos: 0,1).
- Octal, base 8 (dígitos: 0, 1, 2, 3, 4, 5, 6, 7).
- Hexadecimal, base 16 (dígitos: números 0 a 9 y letras A, B, C, D, E, F).

Los sistemas de numeración más usuales son el binario, el decimal, y el hexadecimal; también, y en menor medida, el sistema octal.

Para indicar, explícitamente, que un número es expresado en una determinada base, lo representamos con la base a la que pertenece, como subíndice a la derecha. Por ejemplo:

Base 10: 1234_{10}
Base 16: $23AF_{16}$
Base 2: 110_{2}

En el transcurso del texto, adopta como convención el uso de un sufijo para referirnos a la base a la que pertenece un numero:

Binario: B
Decimal: D (valor por default)
Octal: Q
Hexadecimal: H

Una manera de representar un número en una determinada base es mediante la notación polinómica, de modo que cada dígito, en el sentido de izquierda a derecha, está multiplicado por las sucesivas potencias de la base. Así, por ejemplo, el numero 1234 en base decimal, se puede escribir como:

$$1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

Las computadoras trabajan internamente con el sistema binario, aunque pueden mostrar la información en cualquier otro sistema. En este texto, utilizamos los sistemas binario, decimal y hexadecimal.

Existe una notación formal para expresar cualquier sistema de numeración. Esta notación consiste en indicar, entre paréntesis, la base y la cantidad de dígitos. Así, por ejemplo, un sistema (2,8) hace referencia a un sistema binario, base 2, con hasta 8 dígitos de tamaño.

En general, diremos que un numero pertenece al sistema de numeración (p,q) si está expresado en la base p con q dígitos.

Por ejemplo:

1100 pertenece al sistema (2,4)

AB12 pertenece al sistema (16,4)
093 pertenece al sistema (10,3)

Cuando la cantidad de dígitos significativos de un número que pertenece a un cierto sistema de numeración es inferior a la cantidad q de dígitos del sistema, entonces se deben completar los lugares restantes con ceros a la izquierda.

Por ejemplo:

00101 pertenece al sistema (2,5)
0101 pertenece al sistema (2,4)
000AB pertenece al sistema (16,5)

- **Cambio de base:**

Cualquier número expresado en una base puede, también, ser representado en cualquier otra base. Para poder hacer esto se debe proceder a cambiar la base del número.

En general, para expresar un número en otra base se deben realizar los siguientes pasos:

- Dividir el número a convertir por la nueva base; el resto de la división representa el dígito menos significativo del número en su nueva base.
- Dividir el cociente por la nueva base; el resultado de la división representa el próximo dígito menos significativo del número en su nueva base.
- Seguir así, sucesivamente, hasta que el resultado de la última división sea menor que la nueva base; en este punto, el resto de la última división corresponde al dígito más significativo del número expresado en la nueva base.

Las operaciones que se indican en los pasos descriptos, se deben hacer en la aritmética de la base vieja.

Por ejemplo, para convertir 34_{10} a base 2:

- 34 / 2 = 17 —> resto = 0
- 17 / 2 = 8 —> resto = 1
- 8 / 2 = 4 —> resto = 0
- 4 / 2 = 2 —> resto = 0
- 2 / 2 = 1 —> resto = 0

Entonces $34_{10} = 100010_2$

Los cambios de base más utilizados son: de decimal a binario y viceversa, y de binario a hexadecimal y viceversa.

Para pasar de binario a decimal, se utiliza la forma polinómica; esto es, expresar al número binario descompuesto en potencias de 2 y efectuar las operaciones en aritmética decimal.

Por ejemplo, para convertir el número binario 100010 a base 10:
$1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 32 + 2 = 34_{10}$

Para pasar de binario a hexadecimal, se deben tomar grupos de 4 dígitos binarios justificando, si es necesario, con ceros a la izquierda; y, luego, convertir cada uno de los grupos a un dígito hexadecimal.

Por ejemplo, para convertir el numero binario 11110100101 a base 16:
11110100101 —> 0111 1010 0101 —> $7A5_{16}$

Para pasar de hexadecimal a binario, se convierte cada dígito hexadecimal a su representación binaria, usando siempre cuatro dígitos binarios.

Por ejemplo, la inversa del ejemplo anterior:

$7A5_{16}$ —> $7_{16} = 0111_2$
$A_{16} = 1010_2$
$5_{16} = 0101_2$ —> 011110100101_2

- **Complementación:**

Cuando se trabaja con números en bases diferentes a la decimal, surge el inconveniente de cómo representar a los números negativos.

Esto se resuelve mediante el concepto de complementación, de modo que siempre se representen cantidades positivas –aún cuando su significado puede ser negativo–.

Para implementar este concepto, se define a un número negativo como aquel que, sumándole su positivo, el resultado es 0.

En términos prácticos, trabajaremos con la notación complemento en números binarios, tomando siempre al primer dígito como dígito de signo.

Para complementar un número binario, completamos los siguientes pasos:

- Convertir cada dígito a su inverso (los unos a cero y los ceros a uno).
- Si se utiliza la notación complemento a 2, sumar un 1 al número resultante de la inversión de los dígitos.

Por ejemplo, el complemento de 0101 es $1010 + 1 = 1011$.

Para expresar en forma polinómica un número en notación complemento a 2, se debe tener en cuenta que el dígito más significativo representa el signo.

En el ejemplo: $1011_2 = -1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -8 + 2 + 1 = -5_{10}$

Notar que 1011 (-5_{10}) es el complemento de 0101 (5_{10}).

Representación de la información

Ya hemos visto que existen distintos sistemas de numeración que permiten expresar números en diferentes bases y que la computadora está diseñada para realizar operaciones aritméticas y lógicas con cantidades binarias.

Y, para poder efectuar esas operaciones debe poder, en primera instancia, almacenar esas cantidades numéricas, llamándose **dato** a toda cantidad almacenada en la computadora.

Los datos se almacenan en la memoria de la computadora, en distintas unidades de almacenamiento. Las unidades de almacenamiento más comunes son:

- **Bit.** Abreviatura de la expresión inglesa *Binary Digit.* Es la mínima unidad de almacenamiento y permite representar un único dígito binario; es decir, almacena un uno o un cero. Toda la electrónica digital se basa en el concepto de dígito binario.
- **Byte.** Es un conjunto de bits que conforma la mínima unidad direccionable; cada byte puede ser referenciado mediante un número de orden o dirección dentro de la memoria. En general, el tamaño de un byte es de 8 bits, aunque existen computadoras diseñadas con bytes de otros tamaños. Desde el punto de vista de sistemas de numeración, un byte pertenece al sistema (2,8).
- **Nibble.** Ocupa medio byte; en nuestro caso, un nibble ocupa 4 bits. Es útil para la representación de la información en notación hexadecimal, dado que 4 dígitos binarios equivalen a 1 dígito hexadecimal.
- **Palabra.** Es un conjunto de bits que puede procesarse en forma paralela. El tamaño de una palabra (o *word*) depende de cada máquina aunque, en general, equivale a 2 bytes; en nuestro caso, una palabra tiene 16 bits de tamaño. Desde el punto de los sistemas de numeración, una palabra pertenece al sistema (2,16).

Además de las unidades de almacenamiento indicadas, existen otras, menos frecuentemente usadas: doble palabra, cuádruple palabra, etc.

- **Códigos:**

Para normatizar y facilitar la representación de diferentes símbolos a través de la combinación de dígitos binarios en un byte, se crearon diferentes códigos estándar.

Estos códigos tienen como fundamento la necesidad de poder estandarizar la representación de la información, para permitir que se puedan intercambiar datos entre distintas computadoras.

Los códigos más difundidos, como decíamos a lo largo de las fichas de nuestro texto, son el EBCDIC –*Extended Binary Coded Decimal Interchange Code*– y el ASCII –*American Standard Code Information Interchange*–.

- **EBCDIC.** Este código, especialmente difundido en computadoras grandes –*Main Frames*–, se basa en la codificación de 256 símbolos distintos, entre los que se incluyen todas las letras mayúsculas y minúsculas, los dígitos del 0 al 9, caracteres especiales, etc. Para representar 256 códigos se utilizan 8 bits, de modo que cada símbolo ocupa un byte.
- **ASCII.** Este código es el más ampliamente difundido. De un modo similar al EBCDIC, el ASCII permite representar símbolos. Sin embargo, a diferencia de aquel, sólo admite la codificación de 128 símbolos distintos, pues cada símbolo ocupa los 7 bits menos significativos (7 bits de la derecha) de un byte, quedando el bit 8 como bit de control. No obstante esto, actualmente es muy común encontrar computadoras que incluyen un código ASCII extendido, que se basa en la utilización del bit 8 para poder representar, en total, 256 símbolos distintos (los 128 símbolos originales del ASCII más símbolos definidos con la utilización del bit 8). Los primeros 32 símbolos del código ASCII son los denominados caracteres de control o caracteres no imprimibles. Los códigos de control son especialmente usados en comunicaciones.

- **Tipos de datos:**

Así como hemos visto las diferentes unidades de almacenamiento (bit, byte, palabra,

etc.), es necesario definir los tipos de datos que pueden ser representados en esas unidades de almacenamiento.

Cualquier computadora puede trabajar con datos alfanuméricos y con datos numéricos.

Los datos alfanuméricos son cadenas de caracteres (o string) ASCII o EBCDIC, donde cada carácter ocupa un byte. Así, por ejemplo, la representación de *Hola* en código ASCII será vista en hexadecimal: 48 4F 4C 41.

Los datos numéricos pueden ser:

- binarios,
- decimales desempaquetados,
- decimales empaquetados y
- punto flotante.

Binarios: Son cantidades de uno o más bytes con o sin signo. Para representar binarios con signo, se utiliza la notación complemento a 2. (En el apéndice "Sistemas de numeración" incluimos algunas especificaciones para los números enteros representables en el microprocesador 8088-8086).

Decimales desempaquetados: Se representan a razón de un dígito BCD –*Binary-Coded Decimal*– por byte, que ocupa el nibble más bajo (4 bits menos significativos). Para el 8088-8086 el nibble más alto es siempre cero. Para otras arquitecturas, el nibble más alto está ocupado por un dígito de zona, que en general es 3.

Cada byte puede almacenar un valor entre 0 y 9.

Los dígitos BCD tienen la siguiente codificación:

Dígito BCD		Dígito BCD	
0	0000	**1**	0001
2	0010	**3**	0011
4	0100	**5**	0101
6	0110	**7**	0111
8	1000	**9**	1001

El tamaño de un dato decimal desempaquetado está dado por la cantidad de dígitos que tiene. Por ejemplo, el número 1234 ocupa 4 bytes, y su configuración ser 01 02 03 04.

El microprocesador 8088-8086 opera sólo con números decimales desempaquetados positivos.

Decimales empaquetados: Cada byte contiene dos dígitos BCD. El dígito más significativo se almacena en el nibble más alto y el dígito menos significativo se almacena en el nibble más bajo. Cada byte puede almacenar un valor entre 00 y 99.

El tamaño de un dato decimal empaquetado está dado por la mitad de la cantidad de dígitos que tiene (si un dato tiene una cantidad impar de dígitos, entonces el nibble más alto del dígito más significativo se pone en cero). Por ejemplo, el número 12345 ocupa 3 bytes y su configuración: 01 23 45.

El microprocesador 8088-8086 opera sólo con números decimales empaquetados positivos.

Punto flotante: Los números en punto flotante son representados en una cadena de 4 bytes para simple precisión o de 8 bytes para doble precisión.

Un número en punto flotante tiene 3 campos: signo de la mantisa, exponente y mantisa. (En el apéndice «Sistemas de numeración» incluimos información acerca de los números en punto flotante, en simple y en doble precisión.)

- **Código ASCII:**

Las computadoras deben manejar otros tipos de información además de los números.

Tanto los textos (caracteres alfanuméricos) como las instrucciones deben codificarse de tal modo que la computadora interprete esta información. El código más común para la información tipo texto es el *American Standard Code for Information Interchange* –ASCII–. El código ASCII es una correlación ampliamente aceptada entre caracteres alfanuméricos y valores binarios específicos.

En este código, el número $41 corresponde a una letra A mayúscula, el $20 al carácter espacio, etc. El código ASCII traduce un carácter a un código binario de 7 bits, aunque en la práctica la mayoría de las veces la información es transportada en caracteres de 8 bits con el bit más significativo en cero.

Este estándar permite hace posible las comunicaciones entre equipos hechos por diversos fabricantes, puesto que todas las máquinas utilizan el mismo código.

Tabla de caracteres ASCII:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DEL	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB		7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

NUL - Nulo (Null)
SOH - Comienzo de cabecera (Start of Heading)
STX - Comienzo de texto (Start of Text)
ETX - Final de texto (End of Text)
EOT - Fin de transmisión (End of Transmision)
ENQ - Requerimiento (Enquiry)

ACK - Reconocimiento (Acknowledge)
BEL - Campanilla (Bell)
BS - Retroceso (Back Space)
HT - Tabulación horizontal (Horizontal Tabulation)
LF - Avance de línea (Line Feed)
VT - Tabulación vertical (Vertical Tabulation)
FF - Avance de página (Form Feed)
CR - Retorno de carro (Carriage Return)
SO - Shift Out
SI - Shift In
DLE - Escape de enlace de datos (Data Link Escape)
DCn - Control del dispositivo n (Device Control)
NAK - Reconocimiento negativo (Negative Acknowledge)
SYN - Espera de sincronismo (Synchronous Idle)
ETB - Fin de bloque de transmisión (End of Transmision Block)
CAN - Cancelar (Cancel)
EM - Fin del medio (End of Medium)
SUB - Sustituir (Substitute)
ESC - Escapar (Escape)
FS - Separador de archivo (File Separator)
GS - Separador de grupo (Group Separator)
RS - Separador de registro (Record Separator)
US - Separador de unidad (Unit Separator)
SP - Espacio (Space)
DEL - Borrar (Delete)

- **Binario codificado en decimal:**

El sistema Binario codificado en decimal –BCD– es una notación híbrida, usada para expresar valores decimales en forma binaria. Un BCD utiliza cuatro bits para representar cada dígito decimal.

De esta manera, cuatro dígitos binarios pueden expresar 16 diferentes cantidades físicas, habiendo seis combinaciones consideradas no válidas (específicamente, los valores hexadecimales de la “A” a la “F”). Los valores BCD se representan con el signo “$”, pues ellos son números hexadecimales que representan cantidades decimales.

Cuando la computadora hace una operación de suma BCD, realiza una suma binaria y, luego, realiza un ajuste que genera un resultado BCD.

Como ejemplo, consideremos la siguiente suma BCD:
9(10) + 1(10) = 10 (10) donde (10), significa Base 10.

La computadora suma.....
0000 1001(2) + 0000 0001(2) = 0000 1010(2)
donde (2), significa Base 2.

Pero 1010(2) es equivalente a “A(16)” que es un código BCD no válido.

Cuando la computadora termina el cálculo, realiza un chequeo para ver si el resultado es un código BCD válido. Si hubo un “acarreo” (un desborde) de un dígito BCD a otro o si hubiese algún código no válido, se desencadena una secuencia de etapas para corregir el resultado y llevarlo al formato BCD apropiado. El número 0000 1010(2) es corregido y se transforma en 0001 0000(2) (BCD 10) en este ejemplo.

Decimal	BCD	Binary	Hexadecimal (reference)
0 1 2 3 4 5 6 7 8 9	$0 $1 $2 $3 $4 $5 $6 $7 $8 $9	0000 0001 0010 0011 0100 0101 0110 0111 1000 1001	$0 $1 $2 $3 $4 $5 $6 $7 $8 $9
Combinaciones BCD Inválidas		1010 1011 1100 1101 1110 1111	$A $B $C $D $E $F
10 99	$10 $99	0001 0000 1001 1001	$10 $99

En la mayoría de los casos es ineficiente utilizar la notación BCD para los cálculos de la computadora. Es mejor convertir la información de decimal a binario en el momento de su ingreso, realizar todos los cálculos en binario y convertirlos nuevamente a BCD o decimal sólo si es necesario presentarlos en un exhibidor.

No todos los microcontroladores son capaces de realizar cálculos en BCD, ya que se necesita tener la indicación del acarreo dígito a dígito que no está presente en todas las computadoras (tener en cuenta que en los MCU de Motorola tienen este indicador de semi-acarreo). Forzar a una computadora a comportarse como nosotros necesitamos, resulta menos eficiente que permitirle trabajar en su sistema de numeración natural.

- **Punto flotante:**

El 8088-8086 no puede operar directamente (a nivel hardware) con números en punto flotante (o números reales); para que lo pueda hacer, es necesario que cada lenguaje de programación brinde una interface de software apropiada. En este apéndice, se describen los números en punto flotante tal como los opera el lenguaje BASIC.

Los números en punto flotante pueden ser de simple y doble precisión, según la cantidad de dígitos disponibles para la mantisa: cuantos más dígitos tenga la mantisa, mayor precisión tendrá el número.

Los números en punto flotante tienen tres campos:

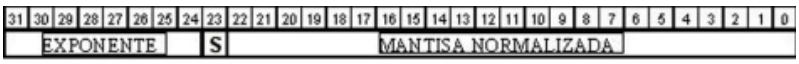
- exponente,
- signo y
- mantisa.

El **exponente** ocupa los primeros 8 bits (byte más alto) y siempre es una cantidad positiva. Para poder representar números con exponente negativo, éste se escribe en notación “con exceso” de 128, de modo que el verdadero exponente de un número es el que resulta de restarle 128 al número representado en el campo de exponente.

El **signo** ocupa el bit que esta a continuación del exponente.

La **mantisa** está a continuación del bit de signo y se encuentra normalizada. Esto significa que, para interpretar la mantisa, se debe agregar un 1 en el bit más alto (al tener la mantisa normalizada, se considera que el primer dígito es siempre un 1 binario aunque, no se lo explicite en la representación). El tamaño de la mantisa varía según la precisión: es de 22 bits para simple precisión y de 51 bits para doble precisión.

Un número de punto flotante en simple precisión ocupa una doble palabra (4 bytes) de memoria y tiene la siguiente estructura:



Bits 0 a 22. Mantisa normalizada. Para ganar precisión se omite representar el primer dígito decimal, pues al estar la mantisa normalizada éste siempre existe. Entonces, al calcular el número se debe considerar este dígito implícito.
Bit 23. Signo de la mantisa. Si es 0, la mantisa es positiva; si es 1, la mantisa es negativa.
Bits 24 a 31. Exponente con exceso a 128. El campo del exponente es un entero positivo; para conocer el valor del exponente se debe restar 128 al valor representado.

El rango para los positivos será entonces:

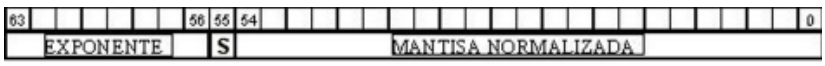
00000000 <= N => FF7FFFFF (hexadecimal)
00000000000000000000000000000000 <= N => 11111111011111111111111111111111 (B)
$0,5 * 2^{-128} <= N => 0,9999999 * 2^{127}$ (exp. base 2)
$2,938736 * 10^{-39} <= N => 1,701412 * 10^{38}$ (exp. base 10)

De la misma manera, el rango para los negativos será:

FFFFFFFF <= N => 00800000 (hexadecimal)
11111111111111111111111111111111 <= N => 00000000100000000000000000000000 (B)
$-0,9999999 * 2^{127} <= N => -0,5 * 2^{-128}$ (exp. base 2)
$-1,701412 * 10^{38} <= N => -2,938736 * 10^{-39}$ (exp. base 10)

De este modo, puede verse que un número positivo difiere del mismo número negativo en el bit de signo.

Un número de punto flotante en doble precisión ocupa 1 cuádruple palabra (8 bytes) de memoria y tiene la siguiente estructura:



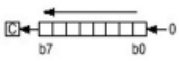
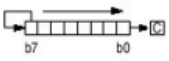
La diferencia con los números de simple precisión es que hay mayor cantidad de dígitos para la mantisa. De este modo, se logra mayor precisión en la representación de un número en punto flotante.

Por ejemplo, el número decimal 2436,55686593 representado en ambos modos da el siguiente resultado:

Simple precisión: 8C1848E9
Doble precisión: 8C1848E8EC3FD9B5

Se nota que, al representar el número en simple precisión, la mantisa se redondea en el último dígito hexadecimal.

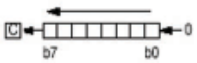
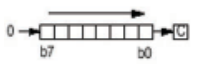
Set de instrucciones de la familia 68HC08

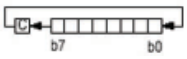
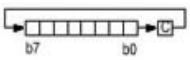
SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
ADC #opr ADC opr ADC opr ADC opr,X ADC opr,X ADC ,X ADC opr,SP ADC opr,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A9 B9 C9 D9 E9 F9 9EE9 9ED9	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ADD #opr ADD opr ADD opr ADD opr,X ADD opr,X ADD ,X ADD opr,SP ADD opr,SP	Add without Carry	A ← (A) + (M)	↕	↕	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
AIS #opr	Add Immediate Value (Signed) to SP	SP ← (SP) + (16 . M)	-	-	-	-	-	-	IMM	A7	ii	2
AIX #opr	Add Immediate Value (Signed) to H:X	H:X ← (H:X) + (16 . M)	-	-	-	-	-	-	IMM	AF	ii	2
AND #opr AND opr AND opr AND opr,X AND opr,X AND ,X AND opr,SP AND opr,SP	LogicalAND	A ← (A) & (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	A4 B4 C4 D4 E4 F4 9EE4 9ED4	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ASL opr ASLA ASLX ASL opr,X ASL ,X ASL opr,SP	Arithmetic Shift Left (Same as LSL)		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR opr ASRA ASRX ARL opr,X ARL opr,X ARL opr,SP	Arithmetic Shift Right		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC rel	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? (C)= 0	-	-	-	-	-	-	REL	24	rr	3
BCLR n,opr	Clear Bit n in M	Mn ← 0	-	-	-	-	-	-	DIR(b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
BCS rel	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + rel ? (C) = 1	-	-	-	-	-	-	REL	25	rr	3
BEQ rel	Branch it Equal	PC ← (PC) + 2 + rel ? (Z) = 1	-	-	-	-	-	-	REL	27	rr	3
BGE opr	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + 2 + rel ? (N & V) = 0	-	-	-	-	-	-	REL	90	rr	3
BGT opr	Branch if Greater Than (Signed Operands)	PC ← (PC) + 2 + rel ? (Z) | (N & V) = 0	-	-	-	-	-	-	REL	92	rr	3
BHCC rel	Branch if Half Carry Bit Clear	PC ← (PC) + 2 + rel ? (H) = 0	-	-	-	-	-	-	REL	28	rr	3
BHCS rel	Branch if Half Carry Bit Set	PC ← (PC) + 2 + rel ? (H) = 1	-	-	-	-	-	-	REL	29	rr	3
BHI rel	Branch if Higher	PC ← (PC) + 2 + rel ? (C) | (Z) = 0	-	-	-	-	-	-	REL	22	rr	3
BHS rel	Branch if Higher or Same (Same as BCC)	PC ← (PC) + 2 + rel ? (C) = 0	-	-	-	-	-	-	REL	24	rr	3
BIH rel	Branch if $\overline{IRQ}$ Pin High	PC ← (PC) + 2 + rel ? $\overline{IRQ}$ = 1	-	-	-	-	-	-	REL	2F	rr	3
BIL rel	Branch if $\overline{IRQ}$ Pin Low	PC ← (PC) + 2 + rel ? $\overline{IRQ}$ = 0	-	-	-	-	-	-	REL	2E	rr	3
BIT #opr BIT opr BIT opr BIT opr,X BIT opr,X BIT ,X BIT opr,SP BIT opr,SP	Bit test	(A) & (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE Opr	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + 2 + rel ? (Z) | (N & V) = 1	-	-	-	-	-	-	REL	93	rr	3
BLO rel	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + rel ? (C) = 1	-	-	-	-	-	-	REL	25	rr	3
BLS rel	Branch if Lower or Same	PC ← (PC) + 2 + rel ? (C) | (Z) = 1	-	-	-	-	-	-	REL	23	rr	3

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
BLT opr	Branch if Less Than (Signed Operands)	PC ← (PC) + 2 + rel ? (N & V) = 1	-	-	-	-	-	-	REL	91	rr	3
BMC rel	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + rel ? (I) = 0	-	-	-	-	-	-	REL	2C	rr	3
BMI rel	Branch if Minus	PC ← (PC) + 2 + rel ? (N) = 1	-	-	-	-	-	-	REL	2B	rr	3
BMS rel	Branch if Interrupt Mask Set	PC ← (PC) + 2 + rel ? (I) = 1	-	-	-	-	-	-	REL	2D	rr	3
BNE rel	Branch if Not Equal	PC ← (PC) + 2 + rel ? (Z) = 0	-	-	-	-	-	-	REL	26	rr	3
BPL rel	Branch if Plus	PC ← (PC) + 2 + rel ? (N) = 0	-	-	-	-	-	-	REL	24	rr	3
BRA rel	Branch Always	PC ← (PC) + 2 + rel	-	-	-	-	-	-	REL	20	rr	3
BRCLR n,opr,rel	Branch if Bit n in M Clear	PC ← (PC) + 3 + rel ? (Mn) = 0	-	-	-	-	-	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN rel	Branch Never	PC ← (PC) + 2	-	-	-	-	-	-	REL	21	rr	3
BRSET n,opr,rel	Branch if Bit n in M Set	PC ← (PC) + 3 + rel ? (Mn) = 1	-	-	-	-	-	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET n,opr	Set Bit n in M	Mn ← 1	-	-	-	-	-	-	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR rel	Branch to Subroutine	PC ← (PC) + 2:push (PCL) SP ← (SP) - 1:push (PCH) SP ← (SP) - 1 PC ← (PC) + rel	-	-	-	-	-	-	REL	AD	rr	4
CBEQ opr,rel CBEQA #opr,rel CBEQX #opr,rel CBEQ opr,X+,rel CBEQ X+,rel CBEQ opr,SP,rel	Compare and Branch it Equal	PC ← (PC) + 3 + rel ? (A) - (M) = $00 PC ← (PC) + 3 + rel ? (A) - (M) = $00 PC ← (PC) + 3 + rel ? (X) - (M) = $00 PC ← (PC) + 3 + rel ? (A) - (M) = $00 PC ← (PC) + 2 + rel ? (A) - (M) = $00 PC ß (PC) + 4 + rel ? (A) - (M) = $00	-	-	-	-	-	-	DIR IMM IMM IX1- IX- SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ii rr rr ff rr	5 4 4 5 4 6

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
CLC	Clear Carry Bit	C ← 0	-	-	-	-	-	0	INH	9B		1
CLI	Clear Interrupt Mark	I ← 0	-	-	0	-	-	-	INH	9A		2
CLR opr CLRA CLRX CLRH CLR opr,X CLR ,X CLR opr,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	-	-	0	1	-	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4
CMP #opr CMP opr CMP opr CMP opr,X CMP opr,X CMP ,X CMP opr,SP CMP opr,SP	Compare A with M	(A) - (M)	↕	-	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A1 B1 C1 D1 E1 F1 9EE1 9ED1	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
COM opr COMA COMX COM opr,X COM ,X COM opr,SP	Complement (One's Complement)	M ← (M) = $FF - (M) A ← (A) = $FF - (M) X ← (X) = $FF - (M) M ← (M) = $FF - (M) M ← (M) = $FF - (M) M ← (M) = $FF - (M)	0	-	-	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5
CPHX #opr CPHX opr	Compare H:X with M	(H:X) ← (M:M + 1)	↕	-	-	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4
CPX #opr CPX opr CPX opr CPX ,X CPX opr,X CPX opr,X CPX opr,SP CPX opr,SP	Compare X with M	(X) - (M)	↕	-	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust	$(A)_{10}$	U	-	-	↕	↕	↕	INH	72		2
DBNZ opr,rel DBNZA rel CBNZX rel DBNZ opr,X,rel DBNZ,X,rel DBNZ opr,SP,rel	Decrement and Branch if Not Zero	A ← (A) - 1 or M = (M) - 1 or X ß (X) - 1 PC ← (PC) + 3 + rel ? (result) ≠ 0 PC ← (PC) + 2 + rel ? (result) ≠ 0 PC ← (PC) + 2 + rel ? (result) ≠ 0 PC ← (PC) + 3 + rel ? (result) ≠ 0 PC ← (PC) + 2 + rel ? (result) ≠ 0 PC ← (PC) + 4 + rel ? (result) ≠ 0	-	-	-	-	-	-	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC opr DECA DECX DEC opr,X DEC ,X DEC opr,SP	Decrement	M ← (M) - 1 A ← (A) - 1 X ← (X) - 1 M ← (M) - 1 M ← (M) - 1 M ← (M) - 1	↕	-	-	↕	↕	-	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)/(X) H ← Remainder	-	-	-	-	↕	↕	INH	52		7

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
EOR #opr EOR opr EOR opr EOR opr,X EOR opr,X EOR ,X EOR opr,SP EOR opr,SP	Exclusive OR M with A	(A) ← (A ⊕ M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
INC opr INCA INCX INC opr,X INC ,X INC opr,SP	Increment	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1 M ← (M) + 1	↕	-	-	↕	↕	-	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP opr JMP opr JMP opr,X JMP opr,X JMP ,X	Jump	PC ← Jump Address	-	-	-	-	-	-	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff 	2 3 4 3 2
JSR opr JSR opr JSR opr,X JSR opr,X JSR ,X	Jump to Subroutine	PC ← (PC) + n (n = 1, 2 or 3) Push (PCL):SP ← (SP) - 1 Push (PCH):SP ← (SP) - 1 PC ← Unconditional Address	-	-	-	-	-	-	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff 	4 5 6 5 4
LDA #opr LDA opr LDA opr LDA opr,X LDA opr,X LDA ,X LDA opr,SP LDA opr,SP	Load A from M	A ← (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LDHX #opr LDHX opr	Load H:X from M	H:X ← (M:M +1)	0	-	-	↕	↕	-	IMM DIR	45 55	ii jj dd	3 4
LDX #opr LDX opr LDX opr LDX opr,X LDX opr,X LDX ,X LDX opr,SP LDX opr,SP	Load X from M X ß (M)	X ← (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	AE BE CE DE EE FE 9EEE 9EDE	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LSL opr LSLA LSLX LSL opr,X LSL ,X LSL opr,SP	Logical Shift Left (Same as ASL)		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
LSR opr LSRA LSRX LSR opr,X LSR ,X LSR opr,SP	Logical Shift Right		↕	-	-	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV opr,opr MOV opr,X+ MOV #opr,opr MOV X+,opr	Move	$(M)_{Destination} \leftarrow (M)_{Source}$ H:X ← (H:X) + 1 (IX + D, DIX+)	0	-	-	↕	↕	-	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
MUL	Unsigned multiply	X:A ← (X) x (A)	-	0	-	-	-	0	INH	42		5
NEG opr NEGA NEGX NEG opr,X NEG ,X NEG opr,SP	Negate (Two's Complement)	M←-(M) = $00 - (M) A←-(A) = $00 - (A) X←-(X) = $00 - (X) M←-(M) = $00 - (M) M←-(M) = $00 - (M)	↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No operation	None	-	-	-	-	-	-	INH	9D		1
NSA	Nibble Swap A	A ← (A [3:0]:A [7:4])	-	-	-	-	-	-	INH	62		3
ORA #opr ORA opr ORA opr ORA opr,X ORA opr,X ORA ,X ORA opr,SP ORA opr,SP	Inclusive OR A and M	A ← (A) | (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	Push A onto Stack	Push (A):SP ← (SP) - 1	-	-	-	-	-	-	INH	87		2
PSHH	Push H onto Stack	Push (H):SP ← (SP) - 1	-	-	-	-	-	-	INH	8B		2
PSHX	Push X onto Stack	Push (X):SP ← (SP) - 1	-	-	-	-	-	-	INH	89		2
PULA	Pull A from Stack	SP ← (SP + 1); Pull (A)	-	-	-	-	-	-	INH	86		2
PULH	Pull H from Stack	SP ← (SP + 1); Pull (H)	-	-	-	-	-	-	INH	8A		2
PULX	Pull X from Stack	SP ← (SP + 1); Pull (X)	-	-	-	-	-	-	INH	88		2
ROL opr ROLA ROLX ROL opr,X ROL ,X ROL opr,SP	Rotate Left through Carry		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR opr RORA RORX ROR opr,X ROR ,X ROR opr,SP	Rotate Right through Carry		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP ← $FF	-	-	-	-	-	-	INH	9C		1
RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP ← SP + 1; Pull (PCH) SP ← SP + 1; Pull (PCL)	-	-	-	-	-	-	INH	81		4

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
SBC #opr SBC opr SBC opr SBC opr,X SBC opr,X SBC ,X SBC opr,SP SBC opr,SP	Subtract with Carry	A ← (A) - (M) - (C)	↕	-	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A2 B2 C2 D2 E2 F2 9EE2 9ED2	ii dd hh ll ee ff ff	2 3 4 4 3 2 4 5
SEC	Set Carry Bit	C ← 1	-	-	-	-	-	1	INH	99		1
SEI	Set Interrupt Mask	I ← 1	-	-	1	-	-	-	INH	9B		2
STA opr STA opr STA opr,X STA opr,X STA ,X STA opr,SP STA opr,SP	Store A in M	M ← (A)	0	-	-	↕	↕	-	DIR EXT IX2 IX1 IX SP1 SP2	B7 C7 D7 E7 F7 9EE7 9ED7	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
STHX opr	Store H:X in M	(M:M + 1) ← (H:X)	0	-	-	↕	↕	-	DIR	35	dd	4
STOP	Enable IRQ Pin Stop Oscillator	I ← 0; Stop Oscillator	-	-	0	-	-	-	INH	8E		1
STX opr STX opr STX opr,X STX opr,X STX ,X STX opr,SP STX opr,SP	Store X in M	M ← (X)	0	-	-	↕	↕	-	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
SUB #opr SUB opr SUB opr SUB opr,X SUB opr,X SUB ,X SUB opr,SP SUB opr,SP	Subtract	A ← (A) - (M)	↕	-	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) - 1; Push (PCH) SP ← (SP) - 1; Push (X) SP ← (SP) - 1; Push (A) SP ← (SP) - 1; Push (CCR) SP ← (SP) - 1; I =1 PCH ← Interrupt Vector High Byte PCH ← Interrupt Vector Low Byte	-	-	1	-	-	-	INH	83		9
TAP	Transfer A to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer A to X	X ← (A)	-	-	-	-	-	-	INH	97		1
TPA	Transfer CCR to A	A ← (CCR)	-	-	-	-	-	-	INH	85		1

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles	Op-code	Cycles
			V	H	I	N	Z	C				
TST opr TSTA TSTX TST opr,X TST ,X TST opr,SP	Test for Negative or Zero	(A) - $00 or (X) - $00 or (M) - $00	0	-	-	↕	↕	-	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to H:X	H:X ← (SP) + 1	-	-	-	-	-	-	INH	95		2
TXA	Transfer X to A	A ← (X)	-	-	-	-	-	-	INH	9F		1
TXS	Transfer H:X to SP	(SP) ← (H:X) - 1	-	-	-	-	-	-	INH	94		2
WAIT	Enable Interrupts; Stop Processor	1 bit ← 0	-	-	0	-	-	-	INH	8F		1

A. Accumulator
C. Carry/borrow bit
CCR. Condition Code Register
dd. Direct Address of Operand
dd cc. Direct Address of Operand and Relative Offset of Branch Instruction
DD. Direct to Direct Addressing Mode
DIR. Direct Addressing Mode.
DIX+. Direct to Indexed with Post Increment Addressing Mode
ee ff. High and Low Bytes of Offset in Indexed, 16-bit Offset Addressing
EXT. Extended Addressing Mode.
ff. Offset Byte in Indexed, o-bit Offset Addressing
H. Half-Carry Biy
H. Index Register High Byte
hh ll. High and Low Bytes of Operand Address in Extended Addressing
I. Interrupt Mask.
ii. Immediate Operand Byte
IMD. Immediate Source to Direct Destination Addressing Mode
IMM. Immediate Addressing Mode
INH. Inherent Addressing Mode
IX. Indexed, no Offset Addressing Mode
IX+. Indexed, no Offset, Post Increment Addressing Mode
IX+D. Indexed with Post Increment to Direct Addressing Mode
IX1. Indexed, 8-bit Addressing Mode
IX1+. Indexed, 8-bit Offset, Post Increment Addressing Mode
IX2. Indexed, 16-bit Offset Addressing Mode
M. Memory Location
N. Negative Bit
n. Any Bit
opr. Operand (one or two bytes)
PC. Program Counter
PCH. Program Counter High Byte
PCL. Program Counter Low Byte
REL. Relative Addressing Mode
rel. Relative Program Counter Offset Byte
rr. Relative Program Counter Offset Byte

SP1. Stack Pointer, 8-bit Offset Addressing Mode
SP2. Stack Pointer, 16-bit Offset Addressing Mode
SP. Stack Pointer
U. Undefined
V. Overflow Bit
X. Index Register Low Byte
Z. Zero Bit
&. Logical AND
|. Logical OR
⊕. Logical EXCLUSIVE OR
( ). Contents of
-( ). Negation (two's Complement)
#. Immediate Value
{{. Sign Extend
←. Loaded with
?. If
:. Concatenated with
↕. Set or Cleared
-. Not Affected

Símbolos de código de condición:
H. Semi acarreo (bit 4)
I. Máscara de interrupción (bit 3)
N. Negativo (bit 2)
Z. Cero (bit 1)
C. Acarreo (bit 0)
0. En bajo
1. En alto
↑. Evalúa y levanta el bit si es verdad (sino, baja el bit)
-. No afectado

Símbolos de expresiones booleanas:
.. AND lógica
:. OR lógica
⊕. OR exclusiva
A. Acumulator
X. Registro índice
M. Posición de memoria

-. Not (inversión)
_. Negativo o resta
+. Suma aritmética
x. Multiplicación
←. Es cargado con...
( ) El contenido de...
CCR. Registro de código de condición
PC. Contador de programa
PCL. PC; byte de menor caso
PCH. PC; byte de mayor peso
SP. Puntero a pila
REL. Desplazamiento relativo

Modo de direccionamiento	Abreviatura	Operandos
Inherente	INH	ninguno
Inmediato	IMM	ii
Directo	DIR	d d
(para evaluación de bits)		dd rr
Extendido	EXT	hh ll
Indexado (sin desplaza-miento)	IX	ninguno
Indexado (con desplaza-miento de 8 bits)	IX1	ff
Indexado (con desplaza-miento de 16 bit)	IX2	ee ff
Relativo	REL	rr

Set de instrucciones PIC 16xxx

Mnemónicos	Op	Descripción	Ciclos	Status
Operaciones de registros orientadas a byte				
ADDWF	f.d	Add W and f	1	C. DC. Z
ANDWF	f.d	AND W weith f	1	Z
CLRF	F	Clear f	1	Z
CLRW	-	Clear W	1	Z
COMF	f.d	Complement f	1	Z
DECF	f.d	Decrement f	1	Z
DECFSZ	f.d	Decrement f. Skip if 0	1 (2)	
INCF	f.d	Increment f	1	Z
INCFSZ	f.d	Increment f. Skip if 0	1 (2)	
IORWF	f.d	Inclusive OR W with F	1	Z
MOVF	f.d	Move f	1	Z
MOVWF	f	Move W to f	1	
NOP	-	No Operation	1	
RLF	f.d	Rotate Left f through Carry	1	C
RRF	f.d	Rotate Right f through Carry	1	C
SUBWF	f.d	Subtract W from f	1	C. DC. Z
SWAPF	f.d	Swap Nibbles in f	1	
XORWF	f.d	Exclusive OR W with f	1	Z
Operaciones de registros orientadas a bit				
BCF	f.b	Bit Clear f	1	1.2
BSF	f.b	Bit Set f	1	1.2
BTFSC	f.b	Bit Test f. Skip if Clear	1 (2)	3
BTFSS	f.b	Bit Test f. Skip if Set	1 (2)	3
Operaciones literales y de control				
ADDLW	k	Add Literal and W	1	C. DC. Z
ANDLW	k	AND Literal with W	1	Z
CALL	k	Call Subroutine	2	
CLRWDT	-	Clear Watchdog Timer	1	TO. PD
GOTO	k	Go to Address	2	
IORLW	k	Inclusive OR literal with W	1	Z
MOVLW	k	Move Literal to W	1	
RETFIE	-	Return from Interrupt	2	
RETLW	k	Return with Literal in W	2	
RETURN	-	Return from Subroutine	2	
SLEEP	-	Go into Standby Mode	1	TO. PD
SUBLW	k	Subtract W from Literal	1	C. DC. Z
XORLW	k	Exclusive OR Literal with W	1	Z

Bibliografía

- Angulo Usategui, J. M.; Angulo Martínez, I. 1999. *Microcontroladores PIC. Diseño práctico de aplicaciones*. Mc Graw Hill.
- Angulo Usategui, J. M.; Martín Cuenca, E.; Angulo Martínez, I. 1997. *Microcontroladores PIC. La solución en un chip.* Paraninfo.
- *Electrónica. Microcontroladores y microprocesadores.* Multipress.
- *Embedded Control Handbook*. 1996. Microchip Technology Inc.
- *MPASM Assembler Quick Reference Guide.* 1996. Microchip Technology Inc.
- *MPSIM Simulator User's guide*. 1996. Microchip Technology Inc.
- *PIC16/17 Microcontroller Data Book.* 1996. Microchip Technology Inc.
- Predko, Myke. 1999. *Programming and Customizing the Pic Microcontroller*. Mc Graw Hill.
- *Technical Training Workbook de Microchip*. 1999. Microchip Technology Inc.

Acerca de microcontroladores Motorola:

- http://mcu.motsps.com/refdesigns
- http://motorola.com/mcu
- latam.msnusers.com/browse.msnw?catid=11&sortby=2
- mx.grulic.org.ar/archiver/html/ grulic/2000-06/msg00532.html
- mx.msnusers.com/68HC08/settings
- mx.msnusers.com/browse.msnw?catid=11&sortby=2
- pegasus.udea.edu.co/Pregrado/ 2001.02/ECNF377-2001-2.rtf
- usuarios.lycos.es/aguzman/documentos/ambiente_ics.pdf
- www.baceelectronica.com.ar/
- www.bairesrobotics.com.ar/data/guia68hc08.pdf
- www.bairesrobotics.com.ar/tutoriales.htm
- www.depeca.uah.es/wwwnueva/docencia/ING-TELECO/sed/
- www.diea.ulpgc.es/docencia/asignaturas/ electronica/14137.html
- www.dsp.efn.uncor.edu/espaniol/info/archivos/ hc08/contador_de_eventos.pdf
- www.dsp.efn.uncor.edu/espaniol/info/pdfs/nanosat.pdf
- www.iearobotics.com/personal/juan/publicaciones/art1/jps.pdf
- www.ii.uam.es/~gdrivera/robotica/curso0203/dia_a_dia.htm
- www.ii.uam.es/~mecatron/documentos/mprog_gp_bot_02.pdf
- www.mundomicro.com.ar/micros/motorola.htm
- www.tic.udc.es/scg/proyect/rinjdael1/proyecto-rinjdael.pdf
- www.upb.edu.co/microprocesadores/capitulo6/cap6_bib.html

Acerca de microcontroladores Intel (Atmel)

- www.8052.com/
- www.atmel.com/products/8051/
- www.cs.ucr.edu/~dalton/i8051/
- www.keil.com/dd/8051chips.asp
- www.pjrc.com/tech/8051/
- www.pjrc.com/tech/8051/ide/

Acerca de microcontroladores PIC:

- Archivos sobre Pics de David Tait. http://www.labyrinth.net.au/~donmck/dtait/index.html
- Bengt Lindgrens HomePage: Programador y archivos. http://home5.swipnet.se/~w-53783

- De todo un poco (Electrónica): Algunos circuitos. http://www.arrakis.es/~ldr2000/manny/circuitos
- Diseño de sistemas con microcontroladores: Enlaces. http://www.info-ab.uclm.es/~amartine
- Dontronics. http://www.dontronics.com
- EDU-PIC: PIC Microcontrollers in education. http://pages.hotbot.com/edu/edu-pic
- El Rincón del Pic. http://members.es.tripod.de/~InfoE/infop.htm
- fae@electrocom.com.ar
- FlashPIC Developer for PIC16F84 and PIC16F87x Series PIC Microcontrollers. http://www.cybermedix.co.nz/flashpic
- Free PIC 16x84 programmer with marginingsupport. http://www.ise.pw.edu.pl/~wzab/picprog/picprog.html
- GNUPic "Free Microcontroller Software Tools http://huizen.dds.nl/~gnupic/index.html
- http://www.arrakis.es/~msyseng
- http://www.geocities.com/CapeCanaveral/Lab/9827/microcon.htm
- http://www.geocities.com/TheTropics/2174/micro.html
- http://www.myke.com/PICMicro
- http://www.pp.clinet.fi/~newmedia/pic/index.html
- Indicadores y controles basados en micros PIC: Otro http://chasque.chasque.apc.org/franky/pics.htm
- La página del autor de Programming and Customizing the Pic Microcontroller: Con algunos circuitos.
- Links sobre PIC de David Tait. http://www.man.ac.uk/~mbhstdj/piclinks.html
- Los Microcontroladores: Información sobre algunos modelos. http://www.gherson.homepage.com
- Microchip Net resources. http://www.geocities.com/SiliconValley/Way/5807
- Microchip. http://www.microchip.com
- Microcontoladores: Información, Herramientas y Programador. Microcontrollers: Enlaces. http://www.us-epanorama.net/microprocessor.html
- Microsystems Engineering: Los autores de los libros de Pics en castellano.
- Página de Javier Alzate: Microcontroladores PIC16CXX.
- Página Web de CX2FW: Información y Links. http://www.angelfire.com/tx/cx2fw/cx2fw.html
- Parallax. http://www.parallaxinc.com
- Pic Programming. Getting Started: 4 pasos para empezar con los Pic.
- PIC16/17 Microcontroller & Basic Stamp: Con algunos proyectos.http://www.doc.ic.ac.uk/~ih/doc/pic
- Proyecto de Gaspar Vidal que utiliza los Pic como soporte hardware. http://www.geocities.com/CapeCanaveral/Campus/8775/proyecto/pfc.htm
- Rei Project: Mod Chip: Algunos proyectos. http://chip.aeug.org
- Sagitron: Distribuidor de Microchip en España. http://www.sagitron.es
- The Electronic Projects Page: Algunos proyectos. http://www.blichfeldt.dk
- The Picmicro Ring. http://members.tripod.com/~mdileo/pmring.html
- The ultimate source for Pic and SX Tools http://www.adv-transdata.com