Microprocesadores y microcontroladores 3

Fichas 5 y 6

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

FICHA 5

Set de instrucciones

El poder de cualquier computadora radica en la habilidad para acceder a la memoria. Son los modos de direccionamiento de la CPU los que proveen esta capacidad.

Los modos de direccionamiento difieren la manera en que una instrucción obtiene el dato requerido para su ejecución.

Debido a los diferentes modos de direccionamiento, una instrucción puede acceder al operando en una de diversas maneras. Cada variante de diferente modo de direccionamiento de una instrucción debe tener un único código de operación de instrucción, de tal modo que las 90 instrucciones básicas de la CPU del MC68HC08 requieren, aproximadamente, 250 códigos de operación de instrucción distintos.

Modos de direccionamiento

La CPU del MC68HC08 usa siete modos de direccionamiento para hacer referencia a memoria:

1. inmediato,
2. inherente,
3. extendido,
4. directo,
5. indexado (sin desplazamiento, con desplazamiento de 8 bits, de 16 bits, con post incremento, o de stack),
6. memoria a memoria y
7. relativo.

En los microcontroladores de la familia MC68HC08, todas las variables del programa y los registros de I/O caben en el área de memoria que va de $0000 a $00FF; allí, el modo de direccionamiento más comúnmente usado es el direccionamiento directo.

Sumario de modos de direccionamiento del 68HC908

Modo	Ejemplo de uso
Inherente	PULX
Inmediato	ADD #$10
Directo	SUB $50
Extendido	SUB $200
Indexado	
- no offset	STA ,X
- 8 or 16 bit offset	LDX $200,X
- post incremento	CBEQ X+,There
- 8 bit offset w/post inc	CBEQ $50,X+,There
Stack Pointer	
- 8 or 16 bit offset	CLR 5,SP
Relativo (PC)	BEQ Here
Memoria a memoria	
- Inmediato a directo	MOV #$00,$A0
- Directo a directo	MOV $18,$F0
- Indexado post inc a directo	MOV X+,$12
- Directo a indexado post inc	MOV $12,X+

Las novedades para esta CPU HC08 son:

- Manipulación del stack (48 nuevas instrucciones).
  - Push / Pull directo de cualquier registro y suma inmediata al stack.

- Modo de direccionamiento relativo para el stack.
- Variables temporales en el stack pueden ser manipuladas directamente, sin tener que ser cargadas, luego, en el acumulador.

- Extensión del index register (7 instrucciones nuevas); permite el uso total del index register de 16 Bits (H: X).
- Movimientos de memoria a memoria (4 nuevas instrucciones).
- Construcciones en Loop –lazo– (12 nuevas instrucciones).
  - Decrement and Branch.
  - Compare and Branch.
- Mejoras en aritmética (1 nueva instrucción).
  - Multiplicación mas rápida (5 ciclos de clks vs 11 en el HC05).
  - División (16bits / 8 bits).
- Soporta operaciones BCD (2 nuevas instrucciones).
  - Decimal adjust accumulator and nibble swap accumulator.
- Soporta compiladores C (4 nuevas instrucciones).
  - Branch condicionales con operadores signados.

1. Modo de direccionamiento inmediato:

En el modo de direccionamiento inmediato, el operando está contenido en el byte inmediato siguiente al código de operación. Este modo es usado cuando un valor o constante conocido al momento de escribir el programa, no cambia durante la ejecución.

Es una instrucción de dos bytes, uno para el código de operación y otro para el byte de dato inmediato. El símbolo **#** indica que el modo de direccionamiento es inmediato, ya que está señalando que el operando es un número, es decir el dato.

Listado del programa de ejemplo:
$E000 A6 02 **LDA #$02** ;Cargar el acumulador con el valor inmediato

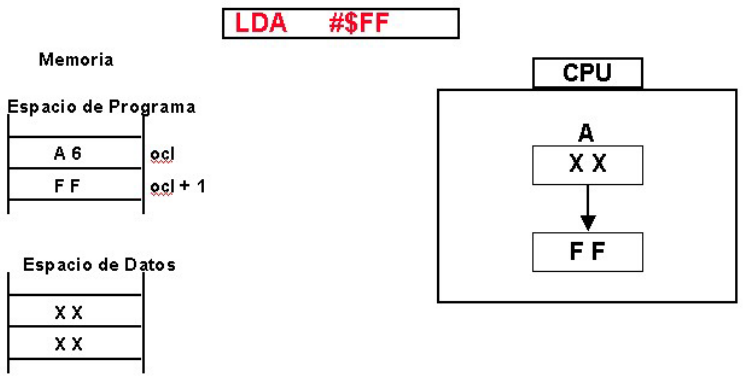
Secuencia de ejecución:
$E000 $A6 [1]
$E001 $02 [2]

Explicación:
[1] La CPU lee el código de operación $A6. Carga del acumulador con el valor inmediato siguiente al código de operación.
[2] La CPU lee el dato inmediato $02 de la posición de memoria $E001 y lo carga en el acumulador.

Especifica el valor directamente, no la dirección del valor
- **Indicado por #**

Tiene un solo operando
- **Contenido en el byte o los bytes seguidos inmediatamente al opcode**

Ejemplo:



La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento inmediato:

MODO DE DIRECCIONAMIENTO INMEDIATO	
INSTRUCCIÓN	MNEMÓNICO
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register and Memory	CPX
Exclusive Or Memory with Accumulator	EOR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Subtract	SUB

2. Modo de direccionamiento inherente:

En el modo de direccionamiento inherente, toda la información requerida para la operación ya es implícitamente conocida por la CPU y no es necesario recuperar un operando exterior desde la memoria.

Los operandos (si los hay) son sólo los registros de la CPU o bien valores de datos almacenados en la pila.

Ésta es una instrucción de un solo byte.

Listado del programa de ejemplo:
$E000 4C **INCA** ; Incrementar el acumulador

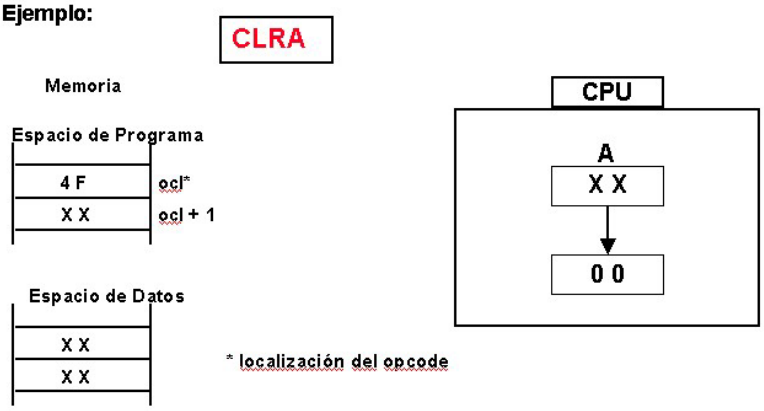
Secuencia de ejecución:
$E000 $4C [1], [2] y [3]

Explicación:
[1] La CPU lee el código de operación $4C. Incremento del acumulador.
[2] La CPU le suma uno al valor actual del acumulador.
[3] La CPU almacena el nuevo valor en el acumulador y ajusta las banderas del registro de código de condición, de ser necesario.

No tiene operando

Mayormente, son operaciones sobre los registros del CPU o bits

Ejemplo:



La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento inherente:

MODO DE DIRECCIONAMIENTO INHERENTE	
INSTRUCCIÓN	MNEMÓNICO
Arithmetic Shift Left	ASLA, ASLX
Arithmetic Shift Right	ASRA, ASRX
Clear Carry Bit	CLC
Clear Interrupt Mask Bit	CLI
Clear	CLRA, CLRX
Complement	COMA, COMX
Decrement	DECA, DECX
Increment	INCA, INCX
Logical Shift Left	LSLA, LSLX
Logical Shift Right	LSRA, LSRX
Multiply	MUL
Negate	NEGA, NEGX
No Operation	NOP
Rotate Left thru Carry	ROLA, ROLX
Rotate Right thru Carry	RORA, RORX
Reset Stack Pointer	RSP
Return from Interrupt	RTI
Return fron Subroutine	RTS
Set Carry Bit	SEC
Set Interrupt Mask Bit	SEI
Enable IRQ. Stop Oscillator	STOP
Software Interrupt	SWI
Transfer Accumulator to Index Register	TAX
Test for Negative or Zero	TSTA, TSTX
Transfer Index Register to Accumulator	TXA
Enable Interrupt. Halt Processor	WAIT

3. Modo de direccionamiento extendido:

En el modo de direccionamiento extendido, la dirección del operando está contenida en los dos bytes siguientes al código de operación.

Este modo es usado para hacer referencia a cualquier posición de memoria dentro del espacio de memoria del MCU, incluyendo I/O, RAM, ROM, EPROM, FLASH.

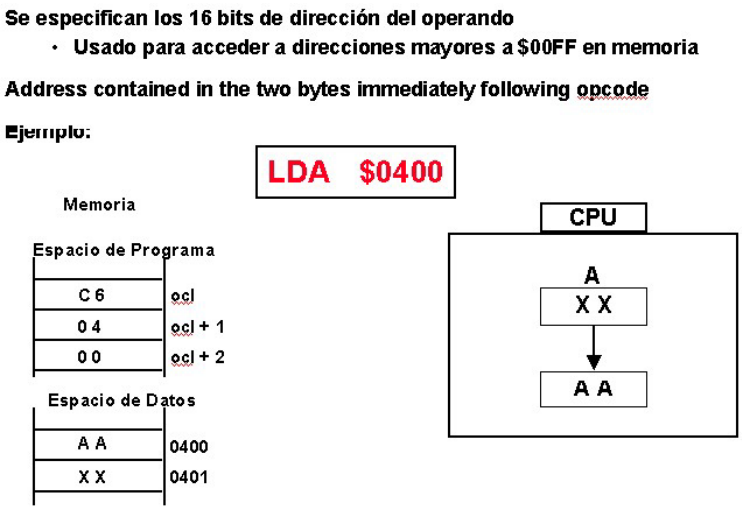
Ésta es una instrucción de tres bytes, uno para el código de operación y otros dos para la dirección del operando.

Listado del programa de ejemplo:
$E000 C6 E3 65 **LDA $E365** ;Cargar el acumulador desde una dirección extendida

Secuencia de ejecución:
$E000 $C6 [1]
$E001 $E3 [2]
$E002 $65 [3] y [4]

Explicación:
[1] La CPU lee el código de operación $C6, Carga del acumulador usando el modo de direccionamiento extendido.

[2] La CPU lee $03 de la posición de memoria $E301. Este $E3 es interpretado como la mitad de mayor peso de una dirección.
[3] La CPU lee $65 de la posición de memoria $E302. Este $65 es interpretado como la mitad de menor peso de una dirección.
[4] La CPU arma la dirección extendida completa $E365 con los dos valores previamente leídos. Esta dirección es colocada en el bus de direcciones; la CPU lee el valor del dato contenido en la posición de memoria $E365 y lo carga en el acumulador.



La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento extendido:

MODO DE DIRECCIONAMIENTO EXTENDIDO	
INSTRUCCIÓN	MNEMÓNICO
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB

4. Modo de direccionamiento directo:

El modo de direccionamiento directo es similar al extendido, excepto que la parte alta de la dirección del operando se asume de valor $00. De tal manera, sólo es necesario incluir el byte de menos peso de la dirección del operando en la instrucción.

Esta área se denomina **página directa** o **página 0** e incluye a los registros de RAM e I/O del interior del chip. Este modo es eficiente, tanto en economía de espacio de memoria de programa como en menor tiempo de ejecución.

Ésta es una instrucción de dos bytes: uno para el código de operación y otro para el byte de menor peso de la dirección del operando.

Listado del programa de ejemplo:
$E000 B6 E0 **LDA $E0** ;Cargar el acumulador desde una dirección de página directa

Secuencia de ejecución:
$E000 $B6 [1]
$E001 $E0 [2] y [3]

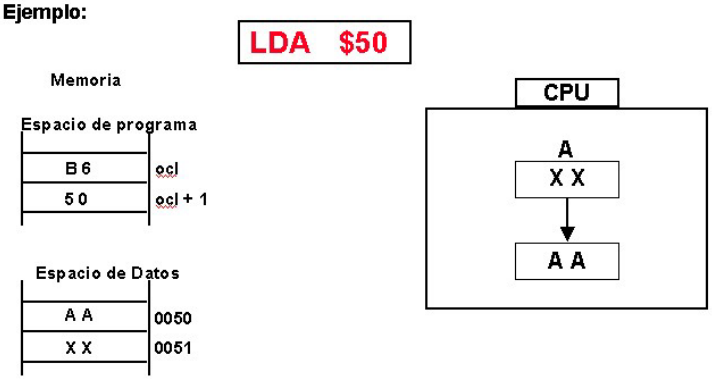
Explicación:
[1] La CPU lee el código de operación $B6. Carga del acumulador usando el modo de direccionamiento directo.
[2] La CPU lee $E0 de la posición de memoria $E301. Este $E0 es interpretado como la mitad de menor peso de una dirección de página directa (desde $0000 hasta $00FF).
[3] La CPU arma la dirección directa completa $00E0 asumiendo el valor del byte de mayor peso en $00, con el valor previamente leído del byte de menor peso. Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en la posición de memoria $00E0 y lo carga en el acumulador.

Se especifican 8 bit de dirección del operando

- **Byte superior de los 16 bits de dirección se asume que sea $00**
- **Usado para acceder a los primeros 256 bytes de memoria**

Dirección contenida en el byte inmediatamente seguido al opcode

Ejemplo:



La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento directo:

MODO DE DIRECCIONAMIENTO DIRECTO	
INSTRUCCIÓN	MNEMÓNICO
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Clear Bit in Memory	BCLR
Bit Test Memory with Accumulator BIT	BIT
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Set Bit in Memory	BSET
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR

Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

5. Modo de direccionamiento indexado:

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

- el contenido actual del registro índice H : X
- el desplazamiento contendido en el/los byte/s siguiente/s al código de operación.

La CPU del MC68HC05 soporta tres tipos de direccionamientos indexados[1]:

5.1. Indexado sin desplazamiento.
5.2. Indexado con desplazamiento de 8 bits.
5.3. Indexado con desplazamiento de 16 bits.

Un buen ensamblador usa el modo de direccionamiento indexado que requiere el menor número de bytes para expresar el desplazamiento.

5.1. Modo de direccionamiento indexado sin desplazamiento:

En el modo de direccionamiento indexado sin desplazamiento, la dirección efectiva del operando para la instrucción está contenida en los 16 bits del registro índice H:X. Así, este modo de direccionamiento puede acceder a las primeras 256 posiciones de memoria (desde $0000 hasta $00FF).

Ésta es una instrucción de un solo byte.

Listado del programa de ejemplo:
$E000 F6 **LDA ,X** ;Cargar el acumulador desde la dirección apuntada por H:X

Secuencia de ejecución:
$E000 $F6 [1], [2] y [3]

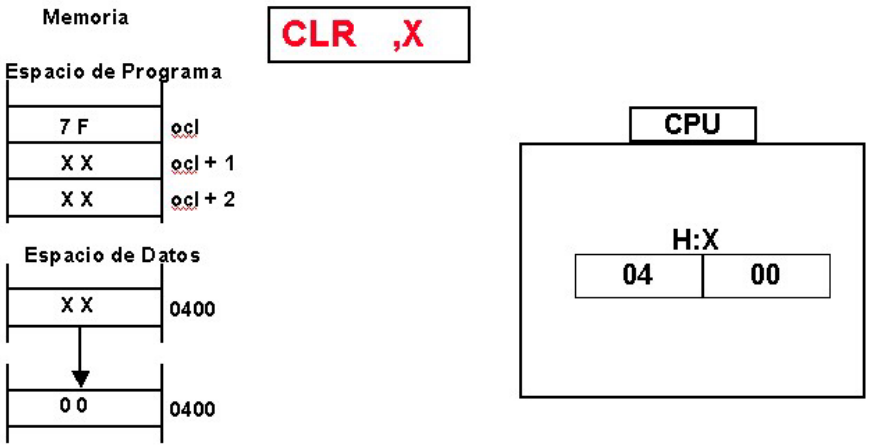
Explicación:
[1] La CPU lee el código de operación $F6. Carga del acumulador usando el modo de direccionamiento indexado sin desplazamiento.

[1] *Además, la CPU 68HC08 incorpora algunos modos de direccionamiento nuevos.*

[2] La CPU arma la dirección completa sumando $0000 al contenido del registro índice de 16 bits H:X.
[3] Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

Se especifica el contenido del index register H:X como dirección del operando

Ejemplo:



La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento indexado sin desplazamiento y con desplazamiento de 8 bits.

MODO DE DIRECCIONAMIENTO INDEXADO SIN DESPLAZAMIENTO Y CON DESPLAZAMIENTO DE 8 BITS

INSTRUCCIÓN	MNEMÓNICO
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Bit Test Memory with Accumulator BIT	BIT
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

5.2. Modo de direccionamiento indexado con desplazamiento de 8 bits:

En el modo de direccionamiento indexado con desplazamiento de 8 bits, la dirección efectiva es la suma del contenido del registro índice de 16 bits H:X y el byte de desplazamiento siguiente al código de operación. El byte de desplazamiento suministrado en la instrucción es un número entero no signado de 8 bits.

Ésta es una instrucción de dos bytes: uno para el código de operación y otro para el byte de desplazamiento. El contenido del registro índice no es alterado.

Listado del programa de ejemplo:

```
$E000 E6 05  LDA 5,X        ;Cargar el acumulador desde el 6º ítem de la tabla apuntada por H:X
```

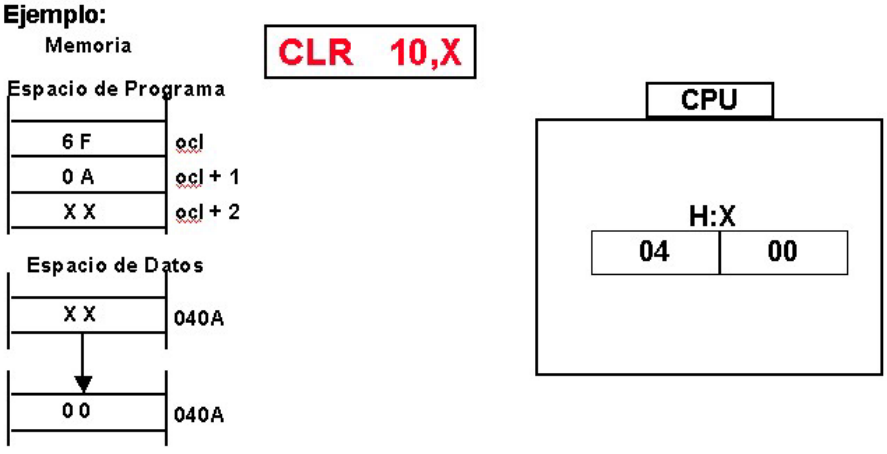
Secuencia de ejecución:
$E000 $E6 [1]
$E001 $05 [2], [3] y [4]

Explicación:
[1] La CPU lee el código de operación $E6. Carga del acumulador usando el modo de direccionamiento indexado con desplazamiento de 8 bits.
[2] La CPU lee $05 de la posición de memoria $0301. Este $05 es interpretado como un desplazamiento de 8 bits.
[3] La CPU arma la dirección completa sumando el valor antes leído ($05) al contenido del registro índice de 16 bits H:X.
[4] Esta dirección es colocada en el bus de direcciones. La CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

8 bit offset no signado + registro H:X no signado = localización de memoria

- **Registro H:X no es afectado**
- **8 bit offset es el byte inmediatamente seguido al opcode**

Ejemplo:



5.3. Modo de direccionamiento indexado con desplazamiento de 16 bits:

En el modo de direccionamiento indexado con desplazamiento de 16 bits, la dirección efectiva es la suma del contenido del registro índice de 16 bits H:X y los dos bytes de desplazamiento siguientes al código de operación.

El byte de desplazamiento suministrado en la instrucción es un número entero no signado de 16 bits.

Ésta es una instrucción de tres bytes: uno para el código de operación y otros dos para los bytes de desplazamiento. El contenido del registro índice no es alterado.

Listado del programa de ejemplo:

```
$E000 D6 E3 77    LDA $E377,X    ;Cargar el acumulador desde el X+1º ítem
                                 de la tabla $E377
```

Secuencia de ejecución:

$E000 $D6 [1]
$E001 $E3 [2]
$E002 $77 [3], [4] y [5]

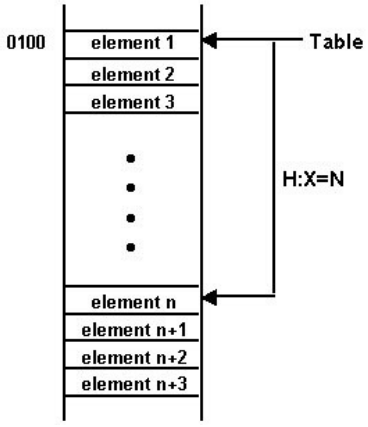
Explicación:

[1] La CPU lee el código de operación $D6. Carga del acumulador usando el modo de direccionamiento indexado con desplazamiento de 16 bits.
[2] La CPU lee $E3 de la posición de memoria $E001. Este $E3 es interpretado como la mitad de mayor peso de una dirección base.
[3] La CPU lee $77 de la posición de memoria $E002. Este $77 es interpretado como la mitad de menor peso de una dirección base.
[4] La CPU arma la dirección completa sumando la dirección base de 16 bits antes leída ($E077) al contenido del registro índice de 16 bits H:X.
[5] Esta dirección es colocada en el bus de direcciones. La CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

Comúnmente usado para acceder a elementos de estructura de datos

- **El Offset sería la dirección base de la estructura**
- **El registro índice contendría el desplazamiento del "N"...avo elemento**

```
            ORG    $100
Table       FCB    $10, $20, $30, $40 .....
            •
            •
* Calcula desplazamiento para elemento N en A
            •
            CLRH
            TAX
            LDA    Table,X
```



NOTA: Si Table estuviera en los primeros 256 bytes de memoria, la mayoría de de los ensambladores usarían instrucciones de 8-bit offset

La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento indexado con desplazamiento de 16 bits.

MODO DE DIRECCIONAMIENTO INDEXADO CON DESPLAZAMIENTO DE 16 BITS	
INSTRUCCIÓN	MNEMÓNICO
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator BIT	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA

Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB

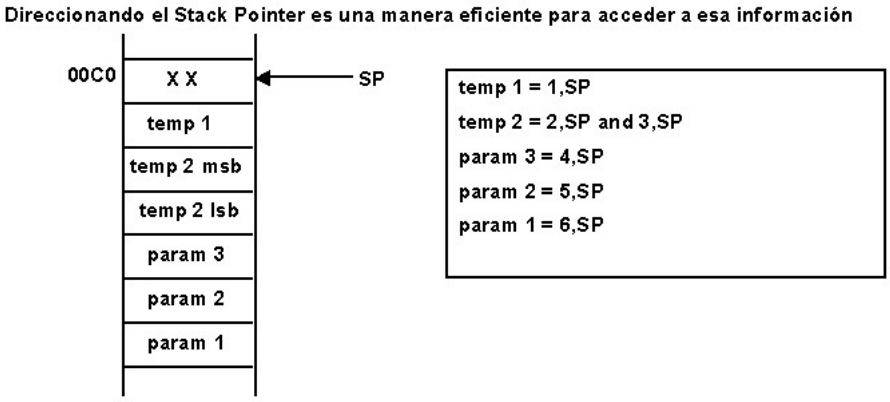
Decíamos que la CPU 68HC08 incorpora algunos modos nuevos de direccionamiento indexado. A continuación los consideraremos:

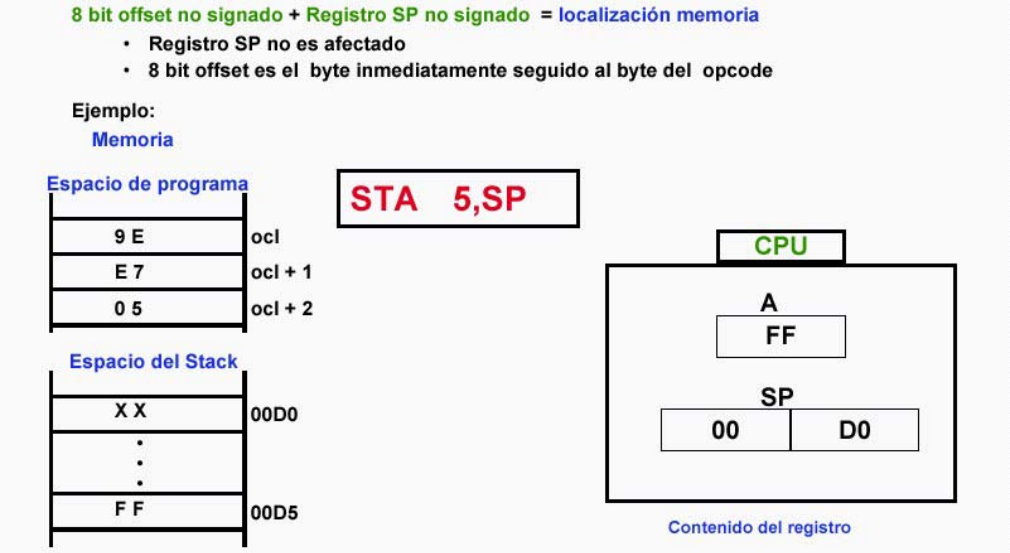
5.4. Indexado usando el stack pointer y desplazamiento de 8 bits (8 bit de offset)
5.5. Indexado usando el stack pointer y desplazamiento de 16 bits (16 bit de offset)
5.6. Indexado no offset con post incremento
5.7. Indexado con 8 bits de offset con post incremento

5.4. Modo de direccionamiento indexado usando el stack pointer y desplazamiento de 8 bits (8 bit de offset):

8 bit offset no signado + Registro SP no signado = localización memoria.

- Registro SP no es afectado.
- 8 bit offset es el byte inmediatamente seguido al byte del opcode.

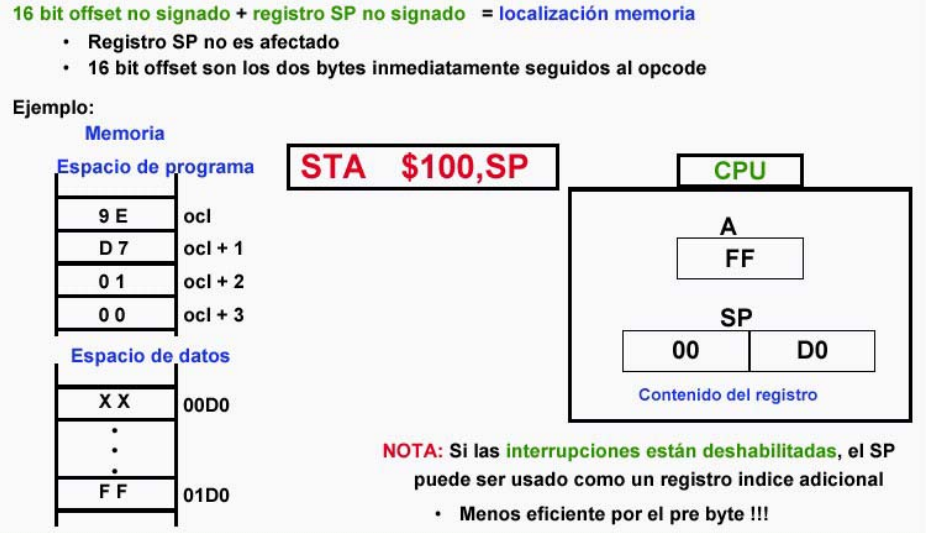




5.5. Modo de direccionamiento indexado usando el stack pointer y desplazamiento de 16 bits (16 bit de offset):

16 bit offset no signado + registro SP no signado = localización memoria

- Registro SP no es afectado.
- 16 bit offset son los dos bytes inmediatamente seguidos al opcode.



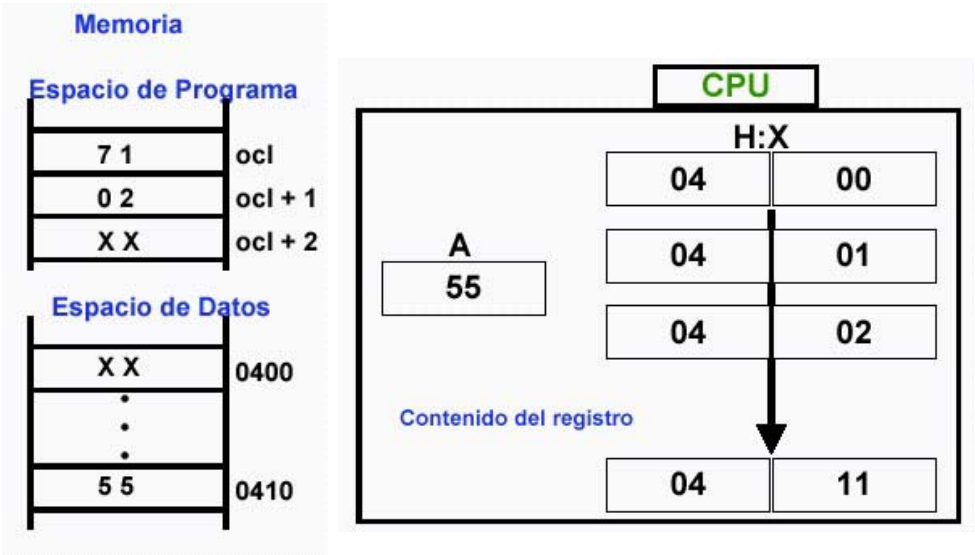
5.6. Modo de direccionamiento indexado no offset con post incremento:

El registro índice H: X contiene la dirección del operando.

Después que la dirección del operando es calculada, H: X es incrementado en 1.

Ejemplo:

```
Loop        CBEQ        X+, Out
            BRA         Loop
Out . . . .
```



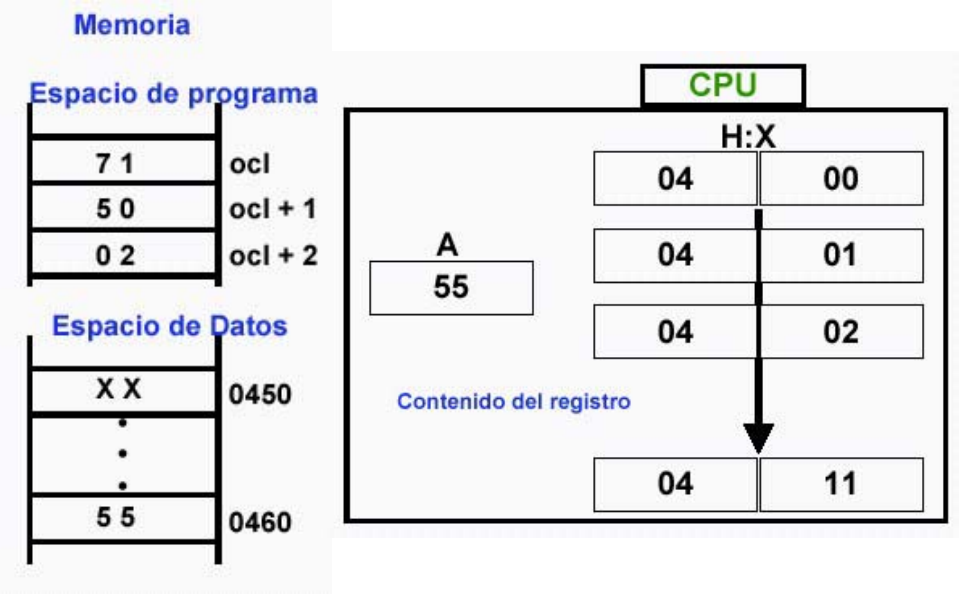
5.7. Modo de direccionamiento indexado con 8 bits de offset con post incremento:

El registro índice H: X contiene la dirección del operando.

Después que la dirección del operando es calculada, H: X es incrementado en 1.

Ejemplo:

```
Loop        CBEQ    $50, X+, Out
            BRA     Loop
Out . . . .
```



6. Modo de direccionamiento memoria a memoria:

Usado para mover información desde una locación a otra:

- No usa/afecta registros del CPU; excepto cuando se usa direccionamiento indexado con post incremento.
- Más eficiente que la combinación Load/ Store.

Pueden utilizarse con intrucciones MOV solamente:

- **MOV Dirección Fuente, Dirección Destino**

El modo de direccionamiento memoria a memoria abarca cuatro variantes:

6.1. inmediato a directo,
6.2. directo a directo,
6.3. indexado a directo con post incremento,
6.4. directo a indexado con post incremento.

6.1. Modo de direccionamiento memoria a memoria, inmediato a directo:

La fuente es un byte valor inmediato. El destino debe estar en los primeros 256 bytes de memoria.

Ejemplo de uso:
Inicialización de variables o registros en RAM:

```
MOV   #$ AA,$ F0
```

6.2. Modo de direccionamiento memoria a memoria, directo a directo:

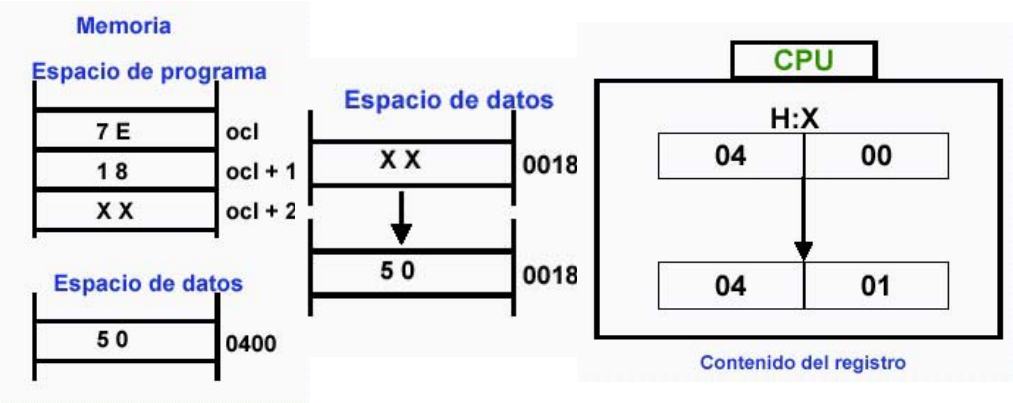
La fuente debe estar en los primeros 256 bytes de memoria. El destino debe estar en los primeros 256 bytes de memoria.

Ejemplo de uso:
Movimiento de datos desde una página cero a otro lugar dentro de la misma página (mover datos dentro de la misma RAM):
MOV $00,$ F0

6.3. Modo de direccionamiento memoria a memoria, indexado con post incremento a directo:

La fuente puede ser cualquier lugar en el mapa de memoria. El destino debe estar en los primeros 256 bytes de memoria

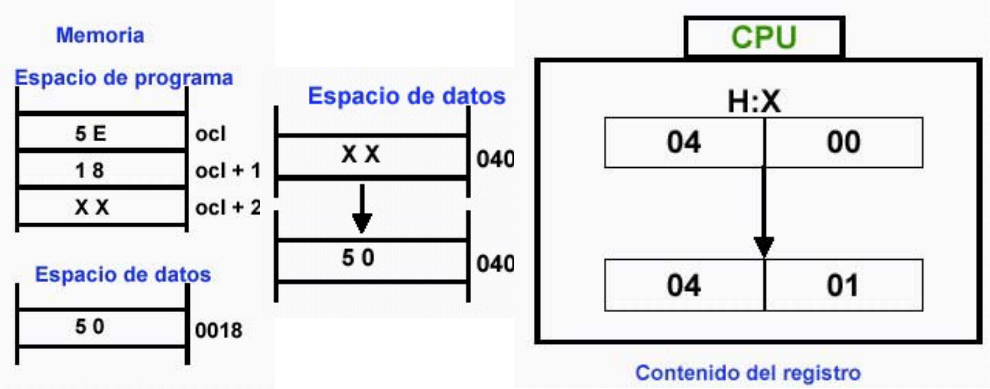
Ejemplo de uso:
Escribir datos a un dispositivo de comunicación desde un buffer en RAM o Flash:
MOV X+,$18



6.4. Modo de direccionamiento memoria a memoria, directo a indexado con post incremento:

La fuente debe estar en los primeros 256 bytes de memoria. El destino puede ser cualquier lugar en el mapa de memoria

Ejemplo de uso:
Escribir datos desde un dispositivo de comunicación a un buffer en RAM o Flash:
MOV $18, X+



7. Modo de direccionamiento relativo:

El modo de direccionamiento relativo es usado solamente por las instrucciones de bifurcación (saltos condicionados). Las instrucciones de bifurcación –salvo las bifurcaciones en su versión de manipulación de bits– generan dos bytes de código de máquina:

- uno para el código de operación y
- otro para el desplazamiento relativo.

Ya que es deseable bifurcar en cualquier sentido, el byte de desplazamiento es un número signado, expresado en complemento a dos, con un rango que va desde -128 hasta +127 bytes (respecto a la dirección de la instrucción inmediata posterior a la instrucción de bifurcación). Si la condición de salto es verdad, el contenido de los 8 bits del byte con signo siguiente al código de operación (desplazamiento) es sumado al contenido del contador de programa para formar la dirección de bifurcación efectiva; de otro modo, el control continúa bajo la instrucción inmediata posterior a la instrucción de bifurcación.

Un programador especifica el destino de una bifurcación como una dirección absoluta (o rótulo que hace referencia a una dirección absoluta). El ensamblador calcula el desplazamiento relativo de 8 bits con signo, que es colocado en memoria luego del código de operación de la bifurcación.

Listado del programa de ejemplo:
$E000 27 rr BEQ DEST ; Bifurcar a DEST si Z = 1 (si es igual o cero)

Secuencia de ejecución:
$E000 $27 [1]
$E001 $ [2] y [3]

Explicación:
[1] La CPU lee el código de operación $27 - Bifurcar si Z = 1. El bit Z del registro de código de condición será uno si el resultado de la operación aritmética o lógica previa fue cero.
[2] La CPU lee $rr de la posición de memoria $E001. Este $rr es interpretado como el valor de desplazamiento relativo. Después de este ciclo, el contador de programa apunta al primer byte de la próxima instrucción ($E002).
[3] Si el bit Z está en cero, nada sucede en este ciclo y el programa debe continuar con la próxima instrucción. Si el bit Z está en uno, la CPU armará la dirección completa sumando el desplazamiento signado antes leído ($rr) al contenido del registro contador de programa para obtener la dirección destino de la bifurcación. Esto provoca que la ejecución del programa continúe desde una nueva dirección (DEST).

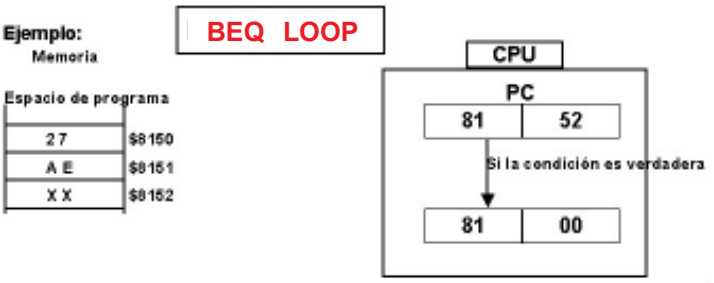
Usado en todas las instrucciones branch condicionales

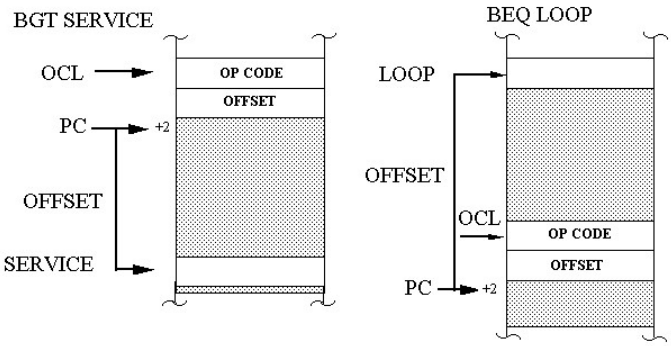
Si la condición es VERDADERA

Program Counter = Program Counter + 8 bit offset signado

SINO

Program Counter no es afectado





- PC is + 2 from OCL (due to prefetching)
- 8 bit offset. Range is -128 +126 from PC
- Effective Address (EA) = PC + displacement(8-bit Offset)
- Assembler calculates displacement: disp. = EA - PC

La tabla incluye una lista de instrucciones que pueden usar el modo de direccionamiento relativo.

MODO DE DIRECCIONAMIENTO RELATIVO	
INSTRUCCIÓN	MNEMÓNICO
Branch if Carry Clear	BCC
Branch is Carry Set	BCS
Branch if Equal	BEQ
Branch if Half Carry Clear	BHCC
Branch if Half Carry Set	BHCS
Branch if Higher	BHI
Branch if Higher or Same	BHS
Branch if Interrupt Line is High	BIH
Branch if Interrupt Line is Low	BIL
Branch if Lower	BLO
Branch if Lower or Same	BLS
Branch if Interrupt Mask is Clear	BMC
Branch if Minus	BMI
Branch if Interrupt Mask Bit is Set	BMS
Branch if Not Equal	BNE
Branch if Plus	BPL
Branch Always	BRA
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Branch Never	BRN
Branch if Subroutine	BSR

Tabla de abreviaturas		
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 desplazamiento)	IX	ninguno
Indexado (con desplazamiento de 8 bits)	IX1	ff
Indexado (con desplazamiento de 16 bits)	IX2	ee ff
Relativo	REL	rr

Clasificación de las instrucciones

El conjunto de instrucciones de un microprocesador define las operaciones básicas que se pueden hacer en una computadora. El microcontrolador tiene un conjunto de 90 instrucciones.

Para aprender el funcionamiento de cada una de ellas, vamos a clasificarlas en los siguientes grupos:

1. Movimiento de datos
2. Aritméticas
3. Lógicas
4. Manipulación de bits
5. Manipulación de datos
6. Control del programa
7. Operaciones BCD
8. Especiales

1. Instrucciones de movimiento de datos:

Consideraremos, dentro de los movimientos o transferencias de datos:

1.1. Carga de registros de CPU
1.2. Almacenamiento de registros del CPU
1.3. Operaciones con el stack
1.4. Registro a registro
1.5. Memoria a memoria

1.1. Movimiento de datos. Carga de registros de CPU:

Este grupo de instrucciones mueve datos entre registros y posiciones de memoria o puertas de E/S. La información que se transfiere es de 8 o de 16 bits.

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		

LDA #opr
LDA opr
LDA opr
LDA opr,X
LDA opr,X
LDA ,X
LDA opr,SP
LDA opr,SP
	Load Accumulator from Memory	A ← (M)	0	-	-	↕[2]	↕	-	IMM
									DIR
									EXT
									IX2
									IX1
									IX
									SP1
									SP2
										2
										3
										4
										4
										3
										2
										4
										5

LDX #opr
LDX opr
LDX opr
LDX opr,X
LDX opr,X
LDX ,X
LDX opr,SP
LDX opr,SP
	Load Index Register X from Memory	X ← (M)	0	-	-	↕	↕	-	IMM
									DIR
									EXT
									IX2
									IX1
									IX
									SP1
									SP2
										2
										3
										4
										4
										3
										2
										4
										5

LDHX #opr LDHX opr	Load Index Register H:X from Memory	H:X ← (M:M + 1)	0	-	-	↕	↕	-	IMM DIR	3 4

1.2. Movimiento de datos. Almacenamiento de registros del CPU:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
STA opr STA opr STA opr,X STA opr,X STA ,X STA opr,SP STA opr,SP	Store Accumulator in Memory	M ← (A)	0	-	-	↕	↕	-	DIR EXT IX2 IX1 IX SP1 SP2	3 4 4 3 2 4 5
STX opr STX opr STX opr,X STX opr,X STX ,X STX opr,SP STX opr,SP	Store Index Register X in Memory	M ← (X)	0	-	-	↕	↕	-	DIR EXT IX2 IX1 IX SP1 SP2	3 4 4 3 2 4 5
STHX opr	Store Index Register H:X in Memory	M:M+1 ← (H:X)	0	-	-	↕	↕	-	DIR	4

[2] ↕ *indica que puede ser seteado o limpiado.*

1.3. Movimiento de datos. Operaciones con el stack:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
PSHA	Push Accumulator onto Stack	Push (A); SP ← (SP - $01)	-	-	-	-	-	-	INH	2
PSHH	Push Index Register H onto Stack	Push (H); SP ← (SP - $01)	-	-	-	-	-	-	INH	2
PSHX	Push Index Register X onto Stack	Push (X); SP ← (SP - $01)	-	-	-	-	-	-	INH	2
PULA	Pull Accumulator from Stack	SP ← (SP + $01); Pull (A)	-	-	-	-	-	-	INH	2
PULH	Pull Index Register H from Stack	SP ← (SP + $01); Pull (H)	-	-	-	-	-	-	INH	2
PULX	Pull Index Register X from Stack	SP ← (SP + $01); Pull (X)	-	-	-	-	-	-	INH	2

1.4. Movimiento de datos. Registro a registro:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
TAP	Transfer Accumulator to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	2
TPA	Transfer CCR to Accumulator	A ← (CCR)	-	-	-	-	-	-	INH	1
TAX	Transfer Accumulator to Index Register X	X ← (A)	-	-	-	-	-	-	INH	1
TXA	Transfer Index Register X to Accumulator	A ← (X)	-	-	-	-	-	-	INH	1
TXS	Transfer Index Register to SP	SPH:SP ← (H:X) - $0001	-	-	-	-	-	-	INH	2
TSX	Transfer SP to Index Register	H:X ← (SPH:SP) + $0001	-	-	-	-	-	-	INH	2

1.5. Movimiento de datos. Memoria a memoria:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
MOV opr,opr	Move	$(M)_{Destination} \leftarrow (M)_{Source}$ H:X ← (H:X) + 1 in X + modes	0	-	-	↕	↕	-	DD	5
MOV opr,X+									DIX+	4
MOV #opr,opr									IMD	4
MOV X+,opr									IX+D	4

2. Instrucciones aritméticas:

Consideraremos:
2.1. Adición
2.2. Sustracción
2.3. Multiplicación y división
2.4. Incremento y decremento
2.5. Complemento y negación
2.6. Comparación
2.7. Misceláneas

2.1. Instrucciones aritméticas. Adición:

Este grupo de instrucciones permite realizar operaciones aritméticas con los distintos tipos de datos numéricos soportados por el microprocesador.

Recuerde usted que el mc maneja números binarios de 8 bits, con o sin signo, y números decimales de 8 bits, empaquetados (2 dígitos BCD por byte) o desempaquetados (1 dígito BCD por byte).

Las instrucciones aritméticas afectan los códigos de condición o banderas de estado; en particular, modifican las banderas de acarreo (Cy), acarreo medio (HC), signo (S), cero (Z) y paridad (P).

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		

ADD #opr
ADD opr
ADD opr
ADD opr,X
ADD opr,X
ADD ,X
ADD opr,SP
ADD opr,SP
	Add Memory to Accumulator without Carry	A ← (A) + (M)	↕	↕	-	↕	↕	↕	IMM
									DIR
									EXT
									IX2
									IX1
									IX
									SP1
									SP2
										2
										3
										4
										4
										3
										2
										4
										5

ADC #opr ADC opr ADC opr ADC opr,X ADC opr,X ADC ,X ADC opr,SP ADC opr,SP	Add Memory to Accumulator with Carry	A ← (A) + (M) + (C)	↕	↕	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5

2.2. Instrucciones aritméticas. Sustracción:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		

SUB #opr
SUB opr
SUB opr
SUB opr,X
SUB opr,X
SUB ,X
SUB opr,SP
SUB opr,SP
	Subtract Memory from Accumulator without Carry	A ← (A) - (M)	↕	-	-	↕	↕	↕	IMM
									DIR
									EXT
									IX2
									IX1
									IX
									SP1
									SP2
										2
										3
										4
										4
										3
										2
										4
										5

SBC #opr SBC opr SBC opr SBC opr,X SBC opr,X SBC ,X SBC opr,SP SBC opr,SP	Subtract Memory from Accumulator with Carry	A ← (A) - (M) - (C)	↕	-	-	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5

2.3. Instrucciones aritméticas. Multiplicación y división:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
MUL	Unsigned 8-bit x 8-bit Multiply	X:A← (X) x (A)	-	0	-	-	-	0	INH	5
DIV	Unsigned 16-bit x 8-bit MDivide	A ← (H:A) ÷ (X) H ← Remainder	-	-	-	-	↕	↕	INH	7

MUL:

- X contendrá el MSB del producto
- A contendrá el LSB del producto

DIV:

- H es el MSB del dividendo
- A es el LSB del dividendo
- X no es afectado

2.4. Instrucciones aritméticas. Incremento y decremento:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
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	4 1 1 4 3 5
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	4 1 1 4 3 5

2.5. Instrucciones aritméticas. Complemento y negación:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
COM opr	Complement (One's Complement)	M ← $FF – (M)	0	-	-	↕	↕	1	DIR	4
COMA		A ← $FF – (A)							INH	1
COMX		X ← $FF – (X)							INH	1
COM opr,X		M ← $FF – (M)							IX1	4
COM ,X		M ← $FF – (M)							IX	3
COM opr,SP		M ← $FF – (M)							SP1	5
NEG opr	Complement (Two's Complement)	M ← $00 – (M)	↕	-	-	↕	↕	↕	DIR	4
NEGA		A ← $00 – (A)							INH	1
NEGX		X ← $00 – (X)							INH	1
NEG opr,X		M ← $00 – (M)							IX1	4
NEG ,X		M ← $00 – (M)							IX	3
NEG opr,SP		M ← $00 – (M)							SP1	5

- Complemento a uno: operación no signada.
- Complemento a dos: operación signada.

2.6. Instrucciones aritméticas. Comparación:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
CMP #opr	Compare Accumulator with Memory	A – (M)	↕	-	-	↕	↕	↕	IMM	2
CMP opr									DIR	3
CMP opr									EXT	4
CMP opr,X									IX2	4
CMP opr,X									IX1	3
CMP ,X									IX	2
CMP opr,SP									SP1	4
CMP opr,SP									SP2	5
CPX #opr	Compare Index Register X with Memory	X – (M)	↕	-	-	↕	↕	↕	IMM	2
CPX opr									DIR	3
CPX opr									EXT	4
CPX opr,X									IX2	4
CPX opr,X									IX1	3
CPX ,X									IX	2
CPX opr,SP									SP1	4
CPX opr,SP									SP2	5
CPHX #opr	Compare Index Register H:X with Memory	H:X – ((M : M+1)	↕	-	-	↕	↕	↕	IMM	3
CPHX opr									DIr	4

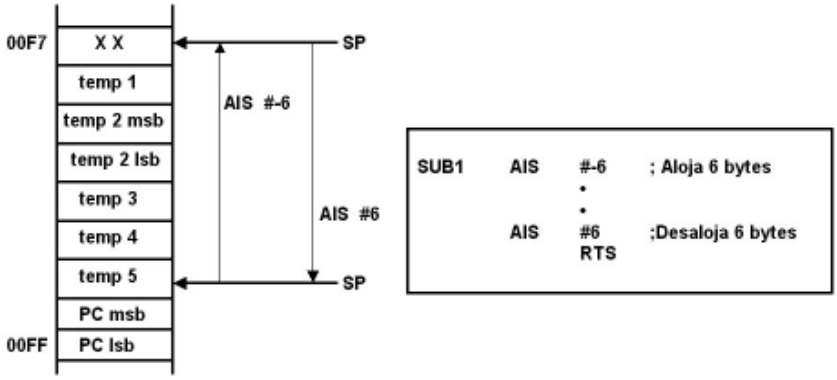
2.7. Instrucciones aritméticas. Misceláneas:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
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	3 1 1 1 3 2 4
TST opr TSTA TSTX TST opr,X TST ,X TST opr,SP	Test for Negative or Zero)	(M) – $00 (A) – $00 (X) – $00 (M) – $00 (M) – $00 (M) – $00	↕	-	-	↕	↕	-	DIR INH INH IX1 IX SP1	3 1 1 3 2 4
AIS #opr	Add Immediate Value (Signed) to Stack Pointer	SHP:SPL ← (SPH:SPL) + (16 << M)	-	-	-	-	-	-	IMM	2
AIX #opr	Add Immediate Value (Signed) to Index Register H:X	H:X ← (H:X) + (16 << M)	-	-	-	-	-	-	IMM	2

Ejemplos:

AIS puede usarse para un rápido alojamiento o desalojo de espacio del Stack

- **Variables Temporales**
- **Procesos en "trama"**



Calcula checksum de 8 bits para una tabla de 512 bytes

```
            ORG   $0100
TABLE       RMB   512          ;Tabla de datos

            ORG   $8000

            LDHX  #511         ;Inicialización del contador de byte

            CLRA               ;Inicializa el  checksum

ADDLOOP     ADD   Table,X      ;Calcula el  checksum
            AIX   #-1          ;Decrementa el contador de  bytes
```

Con DECX no hay “carry” desde X a través de H. AIX si !!!.

```
        CPHX  #0          ;terminó ?
```

PHX setea bits CCR .

```
        BPL   ADDLOOP     ;en Loop si no se completó
```

3. Instrucciones lógicas

Así como las instrucciones aritméticas siempre asumen que sus operandos representan información numérica, las instrucciones de manipulación de bits tratan a los operandos como simples cadenas de bits, lo que permite operar tanto con bits individuales como con máscaras o patrones de bits dentro de registros o de posiciones de memoria.

Las instrucciones lógicas trabajan con máscaras de bits según las reglas de la lógica formal. Se utilizan, básicamente, para la existencia de determinados bits dentro de la máscara.

La siguiente tabla muestra la representación hexadecimal de las distintas máscaras para cada bit individual:

BIT	HEXA
0	0001
1	0002
2	0004
3	0008
4	0010
5	0020
6	0040
7	0080
8	0100
9	0200
10	0400
11	0800
12	1000
13	2000
14	4000
15	8000

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
AND #opr AND opr AND opr AND opr,X AND opr,X AND ,X AND opr,SP AND opr,SP	Logical AND Accumulator and Memory	A ← (A) & (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5
ORA #opr ORA opr ORA opr ORA opr,X ORA opr,X ORA ,X ORA opr,SP ORA opr,SP	Inclusive OR Accumulator and Memory	A ← (A) | (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5
EOR #opr EOR opr EOR opr EOR opr,X EOR opr,X EOR ,X EOR opr,SP EOR opr,SP	Exclusive OR Accumulator and Memory	A ← (A) ⊕ (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5

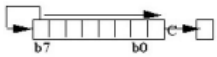
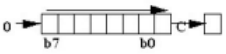
4. Instrucciones de manipulación de bits:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
BIT #opr BIT opr BIT opr BIT opr,X BIT opr,X BIT ,X BIT opr,SP BIT opr,SP	Bit test AND Accumulator with Memory	A & (M)	0	-	-	↕	↕	-	IMM DIR EXT IX2 IX1 IX SP1 SP2	2 3 4 4 3 2 4 5
BCLR n,opr	Clear bit n in Memory	Mn ← 0	-	-	-	-	-	-	DIR	4
BSET n,opr	Set bit n in Memory	Mn ← 1	-	-	-	-	-	-	DIR	4
CLC	Clear Carry Bit	C ← 0	-	-	-	-	-	0	INH	1
SEC	Set Carry Bit	C ← 1	-	-	-	-	-	1	INH	1
CLI	Clear Interrupt Mask	I ← 0	-	-	0	-	-	-	INH	2
SEI	Set Interrupt Mask	O ← 1	-	-	1	-	-	-	INH	2

5. Instrucciones de manipulación de datos:

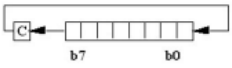
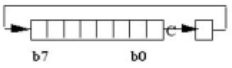
5.1. Manipulación de datos. Shifts:

Estas instrucciones efectúan el desplazamiento de bits, en cantidades de 8 bits (byte) o de 16 bits (palabra), a la izquierda o a la derecha, tantas posiciones como se indiquen. Las posiciones desplazadas se completan con bits en cero.

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
ASL opr ASLA ASLX ASL opr,X ASL ,X ASL opr,SP	Arithmetic Shift Left (Same as LSL)		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	4 1 1 4 3 5
ASR opr ASRA ASRX ASR opr,X ASR ,X ASR opr,SP	Arithmetic Shift Right		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	4 1 1 4 3 5
LSL opr LSLA LSLX LSL opr,X LSL ,X LSL opr,SP	Logical Shift Left		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	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	4 1 1 4 3 5

5.2. Manipulación de datos. Rotaciones:

Las instrucciones de rotación de bits son similares a las de corrimiento de bits; pero, a diferencia de éstas, preservan los bits desplazados ingresándolos al operando por el lado opuesto al del corrimiento.

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
ROL opr ROLA ROLX ROL opr,X ROL ,X ROL opr,SP	Rotate Left through Carry		↕	-	-	↕	↕	↕	DIR INH INH IX1 IX SP1	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	4 1 1 4 3 5

6. Instrucción de control de programa:

6.1. Control de programa. Branches –saltos–:

Las instrucciones de transferencia de control permiten alterar la secuencia normal de ejecución de un programa. Al igual que en los demás lenguajes, las instrucciones de un programa assembler se escriben secuencialmente, es decir una a continuación de la otra. Asimismo, se ejecutan en forma secuencial: el registro de instrucción siempre apunta a la próxima instrucción, siendo ésta la que está a continuación en memoria.

Sin embargo, muchas veces, por una determinada condición, es necesario alterar esa secuencia normal de ejecución de instrucciones, para continuar la ejecución del programa en otra parte. Para esto es que se utilizan las instrucciones de transferencia de control o de bifurcación.

Hay dos tipos de instrucciones de transferencia de control:

- De transferencia de control incondicional.
- De transferencia de control condicional.

En el primer caso, la transferencia de control se hace siempre, sin testear ninguna condición.

En el segundo caso, las instrucciones permiten bifurcar a una parte del programa, siempre que se verifique una condición; si la condición no se satisface, el programa continúa la secuencia normal de ejecución.

La condición puede ser el resultado de una operación aritmética o lógica; pero, siempre se testea basándose en la presencia o ausencia de ciertas condiciones en las banderas de estado.

SourceForms	Description	Operation	Effect on CCR						Address	Bus
			V	H	I	N	Z	C		
BCC rel	Branch if condition is true (CC,CS, HCC, HCS, HI, HS, LO, LS, PL, MI, EQ,Ne, GE, GT, LE, LT, IH,IL,MC,MS)	PC ← (PC) + $0002 + rel ? cc	-	-	-	-	-	-	REL	3
BRA rel	Branch Always	PC ← (PC) + $0002 + rel	-	-	-	-	-	-	REL	3
BRN rel	Branch Never	PC ← (PC) + $0002	-	-	-	-	-	-	REL	3
BRCLR n,opr,rel	Branch if Bit n in Memory is Clear	PC ← (PC) + $0003 + rel ? Mn = 0	-	-	-	-	-	↕	DIR/REL	5
BRSET n,opr,rel	Branch If Bit n in Memory is Set	PC ← (PC) + $0003 + rel ? Mn = 1	-	-	-	-	-	↕	DIR/REL	5

6.2. Control de programa. Saltos especiales:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
CBEQ opr, rel	Compare and Branch if Equal	PC ← (PC) + $0003 + rel ? (A) – (M) = $00	-	-	-	-	-	-	DIR	5
CBEQ A #opr, rel		PC ← (PC) + $0003 + rel ? (A) – (M) = $00							IMM	4
CBEQ X #opr, rel		PC ← (PC) + $0003 + rel ? (X) – (M) = $00							IMM	4
CBEQ X+,rel		PC ← (PC) + $0003 + rel ? (A) – (M) = $00							IX+	4
CBEQ opr,X+,rel		PC ← (PC) + $0002 + rel ? (A) – (M) = $00							IX1+	5
CBEQ opr,SP,rel		PC ← (PC) + $0004 + rel ? (A) – (M) = $00							SP1	6
DBNZ opr, rel	Decrement and Branch if not Zero	M ← (M) - $01PC = (PC) + $0003 + rel ? (M) ≠0	-	-	-	-	-	-	DIR	5
DBNZ A opr, rel		A ← (A) - $01PC = (PC) + $0002 + rel ? (A) ≠0							INH	3
DBNZ X opr, rel		X ← (X) - $01PC = (PC) + $0002 + rel ? (X) ≠0							INH	3
DBNZ X+,rel		M ← (M) - $01PC = (PC) + $0002 + rel ? (M) ≠0							IX	4
DBNZ opr,X+,rel		M ← (M) - $01PC = (PC) + $0003 + rel ? (M) ≠0							IX1	5
DBNZ opr,SP,rel		M ← (M) - $01PC = (PC) + $0004 + rel ? (M) ≠0							SP1	6

CBEQ combina las instrucciones CMP y BEQ

- Operaciones más rápidas de búsqueda/ acceso a tablas.

DBNZ combina las instrucciones DEC y BNE

- Loop's más rápidos y eficientes.

Por ejemplo:

*Subrutina que busca en un string el próximo carácter en blanco
* y luego apunta el registro H:X al carácter inmediatamente
* seguido al blanco.

- H:X se asume que siempre apunta a la localización del string.

```
            ORG   $00A0
String      RMB   50          ; string


            ORG   $8000
Search      LDA   #$20        ;carga carácter de búsqueda (espacio en blanco)

Loop        CBEQ  X+,Out      ;encontrado ?, si, retorno
            BRA   Loop        ;no encontrado entonces vuelvo.

Out         RTS
```

*El post incremento de X ocurrirá independientemente por donde tome el Branch. Por otra parte, cuando una coincidencia es encontrada H:X estará listo para apuntar próximo carácter.

Otro ejemplo:

```
* Rutina de retardo tiempo
* Delay = N x (160.0+0.375) μs para 8 MHz CPU clock
* por ejemplo, para retardo delay = 10ms N = 63

N           EQU   63           ;Loop counter for 10 ms delay

            ORG   $50
Count       RMB   1            ;Loop counter

            ORG   $6E00
Delay       LDA   #N           ;Set delay constant
Loop        DBNZ  Count,Loop   ;Inner loop, Count starts at $00
            DBNZA Loop
            RTS
```

6.3. Control de programa. Jumps –saltos– y subrutinas:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
JMP opr JMP opr JMP opr,X JMP opr,X JMP ,X	Jump to location	PC ← Jump Address	-	-	-	-	-	-	DIR EXT IX2 IX1 IX	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	4 5 6 5 4
BSR rel	Branch to subroutine	PC ← (PC) + 2 Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← (PC) + rel	-	-	-	-	-	-	REL	4
RTS	Return from subroutine	SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	-	-	-	-	-	-	INH	4

6.4. Control de programa. Interrupciones:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
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 PCL ← Interrupt Vector Low Byte	-	-	1	-	-	-	INH	9
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	7

7. Instrucciones de operaciones BCD –Decimal codificado en binario–:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
DAA	Decimal Adjust Accumulator	$(A)_{10}$	U	-	-	↕	↕	↕	INH	2
NSA	Nibble Swap contents of Accumulator	A ← (A [3:0] : A [7:4])	-	-	-	-	-	-	INH	3

8. Instrucciones especiales:

SourceForms	Description	Operation	Effect on CCR						Address Modes	Bus Cycles
			V	H	I	N	Z	C		
RSP	Reset Stack Pointer	SPL ← $FF	-	-	-	-	-	-	INH	1
NOP	No Operation	None	-	-	-	-	-	-	INH	1
STOP	Stop Processor and Wait for Interrupt	I ← 0 Stop Oscillator	-	-	0	-	-	-	INH	1
WAIT	Halt Processor and Wait for Interrupt	I ← 0	-	-	0	-	-	-	INH	1

WAIT:

- El CPU08 detiene el procesamiento de instrucciones.
- Espera por una interrupción.

STOP:

- El CPU08 detiene el procesamiento de instrucciones.
- Detiene el circuito del oscilador.
    - Pone al MPU en estado “low power”.
- Espera por una interrupción.

FICHA 6

Procesando excepciones

En esta última ficha de **Microprocesadores y microcontroladores**, consideraremos el procesamiento de:

- Reset e interrupciones
- Vinculación con el mundo exterior

Reset e interrupciones

Reset e interrupciones son excepciones al proceso normal del CPU.

El procesado de la excepción es manejado a través de tareas discretas:

- Reconocimiento.
- Arbitraje.
- *Stacking* –apilado–.
- Vectorizado.
- *Masking* –enmascaramiento–.

	Reset	**Interrupción**
Reconocimiento –detección de reset o interrupciones pendientes–	Es reconocida y ejecutada inmediatamente, una vez ingresada.	Es reconocida durante el último ciclo de la instrucción corriente. Si ingresa en el último ciclo, entonces es reconocida durante el último ciclo de la próxima instrucción. Actúa después del último ciclo de la instrucción en curso.
Arbitraje	Más alta prioridad. No posee arbitraje.	Difiriendo prioridades. Más bajas que los resets. Realizado por el SIM –*Reset Status Register*–.
Stacking –apilado; salvado de información del CPU–	No se realiza stacking. Se resetea el estado del CPU.	Stacks (apila) registros del CPU: PC, X, A, CCR. El registro H no es apilado, para mantener la compatibilidad con la CPU HC05.
Vectorizado –de excepciones–	Todas usan el mismo vector. Puede determinarse examinando un registro del SIM –*Reset Status Register*–.	El vector depende de la fuente de interrupción.
Masking –enmascaramiento; Habilitación/deshabilitación del procesamiento de excepciones–	No puede ser enmascarado.	Puede ser enmascarada. El "i" bit habilita/deshabilita todas las interrupciones a procesar. Existen máscaras locales en periféricos, que permiten enmascarar interrupciones individualmente.

Vectores de interrupción – CPU 68HC08		
Low Priority	$FFDC	Timebase module vector (high)
	$FFDD	Timebase module vector (low)
	$FFDE	ADC conversion complete vector (high)
	$FFDF	ADC conversion complete vector (low)
	$FFE0	Keyboard vector (high)
	$FFE1	Keyboard vector (low)
	$FFE2	SCI transmit vector (high)
	$FFE3	SCI transmit vector (low)
	$FFE4	SCI receive vector (high)
	$FFE5	SCI receive vector (low)
	$FFE6	SCI receive error vector (high)
	$FFE7	SCI receive error vector (low)
	$FFE8	Reserved
	$FFE9	Reserved
	$FFEA	Reserved
	$FFEB	Reserved
	$FFEC	Reserved
	$FFED	Reserved
	$FFEE	Reserved
	$FFEF	Reserved
	$FFF0	Reserved
	$FFF1	Reserved
	$FFF2	TIM overflow vector (high)
	$FFF3	TIM overflow vector (low)
	$FFF4	TIM channel 1 vector (high)
	$FFF5	TIM channel 1 vector (low)
	$FFF6	TIM channel 0 vector (high)
	$FFF7	TIM channel 0 vector (low)
	$FFF8	CMIREQ vector (high)
	$FFF9	CMIREQ vector (low)
	$FFFA	IRQ1 vector (high)
	$FFFB	IRQ1 vector (low)
	$FFFC	SWI vector (high)
	$FFFD	SWI vector (low)
High Priority	$FFFE	Reset vector (high)
	$FFFF	Reset vector (low)

a) Interrupciones

Las Interrupciones suspenden el procesamiento normal, para que el CPU pueda realizar algunos servicios requeridos.

Son fuentes de Interrupciones:

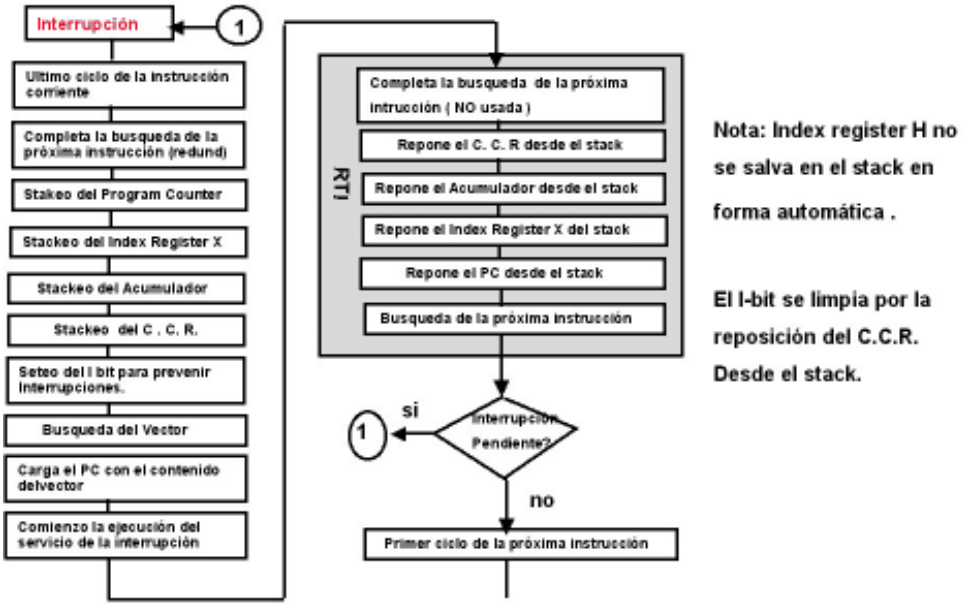
- IRQ external PIN
- IRQ / KBI
- SCI
- SPI
- TIM
- CGM (PLL)
- ADC
- SWI

Son, a veces, usadas para interrumpir el procesamiento normal y, así, responder a algún evento inusual. La CPU MC68HC908 puede ser interrumpida por las siguientes fuentes:

- Un cero lógico aplicado al pin de interrupción externa (IRQ).
- Un cero lógico aplicado a cualquiera de los pines del puerto PA (si la función de puerto de interrupción es habilitada).
- Un pedido de desborde o interrupción de timer (overflow TOF).

- La instrucción de interrupción por programa (SWI).
- Fin de conversión del sistema conversor analógico digital.
- Buffer lleno en el sistema de recepción de datos serial sincrónico SCI.
- Buffer vacío en el sistema de transmisión de datos serial sincrónico SCI.
- Si están habilitadas las interrupciones por ruido, sobreescritura, error de trama o paridad en el sistema de comunicación serial sincrónico.
- Un pedido de comparación en los registros de uno de los dos temporizadores, para salida de comparación o entrada de captura de datos.

El flujograma de funcionamiento de la interrupción:



Si una interrupción se produce mientras la CPU está ejecutando una instrucción, la instrucción es completada antes que la CPU responda al pedido de interrupción.

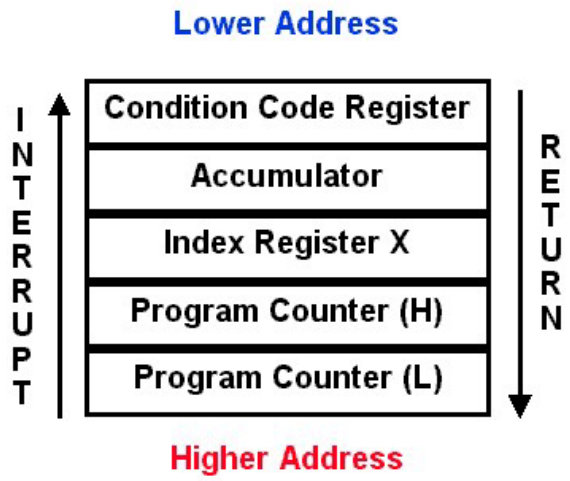
Las interrupciones pueden ser inhibidas:

- en conjunto, poniendo un uno en el bit I del CCR; o bien
- individualmente, poniendo ceros en los bits de control de habilitación de cada fuente de interrupción.

El reset fuerza el bit I a uno y a cero a todos los bits de habilitación de interrupciones locales, a fin de prevenir interrupciones durante el proceso de inicialización. Cuando el bit I está en uno, ninguna interrupción (excepto SWI) es reconocida; aunque puede registrarse la fuente de interrupción, su pedido no es atendido hasta que el bit I se ponga en cero.

Las interrupciones provocan que los registros del procesador sean salvados en la pila y la máscara de interrupciones (el bit I) se ponga en uno, para prevenir interrupciones adicionales hasta la finalización de la presente interrupción. El vector de interrupción apropiado, entonces, apunta a la dirección de inicio de la rutina de atención de interrupción. Completada la rutina de atención de interrupción, una instrucción RTI –que, normalmente, es la última instrucción de una rutina de atención de interrupción– provoca que el contenido de los registros sea recuperado de la pila. De esta forma, el contador de programa es cargado con el valor previamente salvado en la pila, continuando con el procesamiento desde donde nos sacó la interrupción.

En la figura se presenta qué registros son recuperados de la pila, en orden inverso al que fueron salvados.



b) Reset

Los resets inicializan al CPU a un estado conocido.

Todos los resets son manejados por medio del módulo SIM –*System Integration Module*–.

Son tipos de resets:

- Power On
- COP
- Ilegal Address
- External Pin
- LVI
- Ilegal Opcode

El reset es usado para forzar al sistema de MCU a ir a un punto de partida conocido (dirección). Los sistemas periféricos, y muchos bits de control y estado son también forzados a un estado conocido como resultado del reset.

Las siguientes acciones internas ocurren como resultado de cualquier reset del MCU:

1) Todos los registros de dirección de datos se colocan en cero (como entradas).
2) El puntero a la pila (SP) es forzado a $00FF.
3) El bit I del CCR se pone en uno inhibiendo a las interrupciones enmascarables.
4) El latch de interrupciones externas es borrado.
5) El latch de STOP es borrado.
6) El latch de WAIT es borrado.

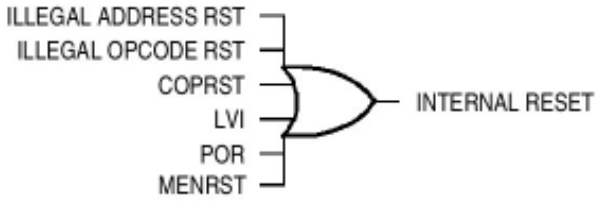
Cuando el sistema de computadora sale de reset, el contador de programa se carga con el contenido de las posiciones de memoria más altas ($FFFE y $FFFF para el MC68HC908). Entonces:

- el valor que se encuentra en $FFFE se carga en el byte más significativo del PC y
- el valor que se encuentra en $FFFF se carga en el byte menos significativo del PC.

Esto se denomina **búsqueda del vector de reset**. En este punto, la CPU comienza la búsqueda y ejecución de instrucciones, comenzando por la dirección almacenada en el vector de reset.

Las siguientes condiciones pueden causar el reset del MC68HC908:

- Externamente, una señal de entrada activa baja en el pin / RESET.
- Internamente, al encender la fuente de alimentación –*power-on reset*; POR–.
- Internamente, expiración de tiempo del cronómetro de vigilancia del comportamiento apropiado de la computadora –*computer operating properly* COP *watchdog timed out*–.
- Un intento de ejecutar una instrucción desde una dirección ilegal.
- Por bajo nivel de tensión de alimentación –LVI–.



- **Pin de reset.** Una llave o un circuito externo puede conectarse a este pin para permitir el reset manual del sistema.

- **Reset al encender la fuente de alimentación –*power-on reset*; POR–.** Al encender la fuente de alimentación, el reset ocurre al detectarse una transición positiva sobre VDD. Su uso es, estrictamente, para la condición de encendido y no podrá utilizarse para detectar caídas de la tensión de la fuente de alimentación. Podrá usarse un circuito inhibidor de baja tensión (LVI) para detectar caídas de la fuente.

  El circuito de power-on provee una demora de 4064 ciclos, desde el momento en que el oscilador se ha activado. Si el pin de /RESET exterior permanece en bajo, al expirar el tiempo de los 4064 ciclos de demora, el procesador permanece en la condición de reset hasta que /RESET se coloque en alto.

- **Reset por *watchdog timer*.** El sistema de cronómetro de vigilancia del comportamiento apropiado de la computadora se propone detectar errores de programas. Cuando se activa el COP, es responsabilidad del programa evitar que un cronómetro de vigilancia que corre libremente llegue al final de su cuenta (Si llega a completar su cuenta, sería una indicación de que el programa no ha sido ejecutado por un largo período de tiempo en la secuencia deseada; entonces, se inicia el reset del sistema).

  Un bit de control del registro de configuración puede usarse para habilitar o deshabilitar el reset del COP. Si el COP es habilitado, la adecuada operación del programa periódicamente debe escribir un cero en el bit COPC del registro de control COPR e ir a la hoja de datos del MC69HC908 por información sobre el ritmo del tiempo de expiración.

  Escribiendo cualquier valor en la dirección $FFFF, previene un COP reset por limpieza del COP contador

- **Reset por acceso a dirección ilegal o código de operación ilegal.** Si el programa es escrito incorrectamente, es posible que la CPU intente saltar o bifurcar a una dirección en la que no haya memoria. Si esto sucede, la CPU podría continuar leyendo datos (resultando ser valores impredecibles) e intentaría actuar en consecuencia, si se tratase de programa. Estas instrucciones sin

sentido pueden provocar que la CPU escriba datos inesperados en memoria o registros diseccionados inesperados. Esta situación se llama **desbocamiento**.

En el MC68HC908 hay un circuito detector de direcciones ilegales para protegernos de esta condición de desbocamiento. Si la CPU trata de buscar una instrucción de una dirección que no pertenece a la memoria FLASH, ni a la ROM monitor, se genera un reset que obliga al programa a comenzar nuevamente.

Vinculación con el mundo exterior

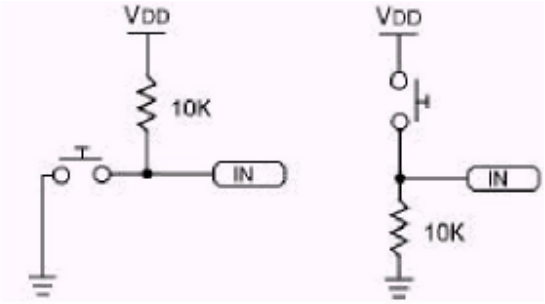
- **Puertos de E/S digitales.** Todos los microcontroladores destinan algunas de sus patitas a soportar líneas de E/S digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando puertos. Las líneas digitales de las puertos pueden configurarse como Entrada o como Salida, cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.

- **Periféricos de E/S.** Los diseños reales utilizan diversos periféricos que hay que conectar a las patillas del microcontrolador que soportan las líneas de E/S.

- **Periféricos digitales de entrada:** Pulsadores, interruptores.

  Pulsadores. Estos dispositivos permiten introducir un nivel lógico en el momento en que se los acciona, pasando al nivel contrario cuando se deja de hacerlo (vuelven a la posición de reposo).

  En el esquema, la línea de entrada (IN) recibe un nivel lógico alto cuando el pulsador está reposo y un nivel lógico bajo cuando se acciona. El pulsador de la derecha funciona al revés.

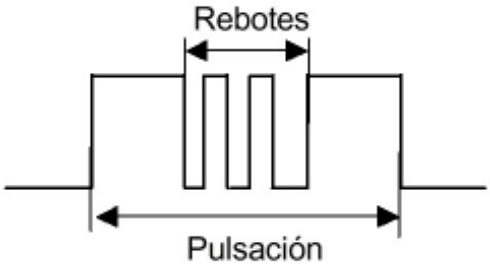


Dos posibles formas de conectar un pulsador

  Hay multitud de detectores, finales de carrera y sensores digitales que funcionan de la misma manera que los pulsadores.

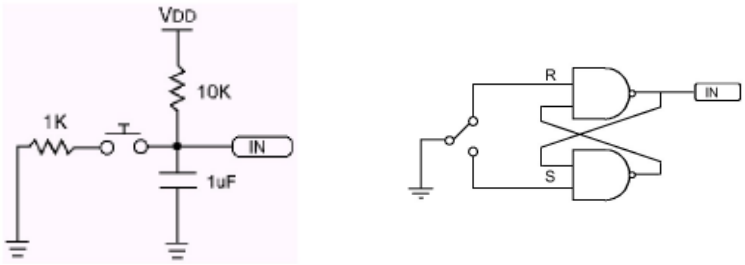
  Interruptores. Los interruptores tienen dos estados estables; hay que accionarlos para cambiar de uno a otro. El interruptor admite el estado *abierto* y el estado *cerrado*. Las formas de conectar un interruptor a una entrada del microcontrolador son iguales a las de la figura anterior, sustituyendo el pulsador por el interruptor.

  Todos los circuitos electromecánicos (pulsadores, interruptores...) originan un fenómeno denominado rebotes: las láminas se abren y se cierran varias veces en el momento de la transición



El efecto que produce es semejante a abrir y cerrar el interruptor o pulsador varias veces, por lo que puede provocar resultados erróneos.

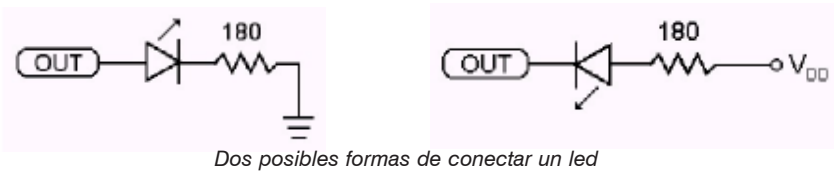
El efecto de los rebotes se puede solucionar mediante software o por hardware. En la figura se muestran dos circuitos hardware antirrebotes. El circuito de la izquierda emplea un condensador y el de la derecha un flip-flop R-S.



Esquemas para eliminar rebotes

- **Periféricos digitales de salida:** Diodos LED, relés

Diodos LED. El diodo led es un elemento que se emplea como indicador luminoso. Cuando la diferencia de potencial entre su ánodo y su cátodo supera un determinado valor umbral, el diodo led se enciende. Los micro-controladores pueden suministrar suficiente corriente como para encender a un diodo led, por eso se pueden conectar directamente a través de una resistencia –como muestra la figura–. Si empleamos la conexión de la izquierda de la figura, el diodo led se enciende al poner a '1' la salida del microcontrolador; con la conexión de la derecha, lo hace cuando la salida se pone a '0'.

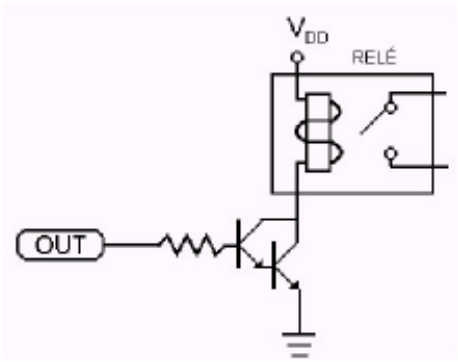


Dos posibles formas de conectar un led

En ocasiones, los diodos u otro tipo de carga necesitan más corriente que la que pueden entregar los microcontroladores. En ese caso, es necesario intercalar una etapa amplificadora.

Relés. La activación y desactivación de un relé brinda la oportunidad de poder controlar cargas mucho mayores (más corriente) porque pueden ser controladas por los contactos de dicho relé. Cuando la línea de salida, OUT, aplica un nivel alto a la base del transistor Darlington (etapa

amplificadora), hace que conduzca y se active el relé. Al cerrarse los contactos del relé se controla una carga mayor. El valor de la resistencia depende del tipo de relé y del transistor.



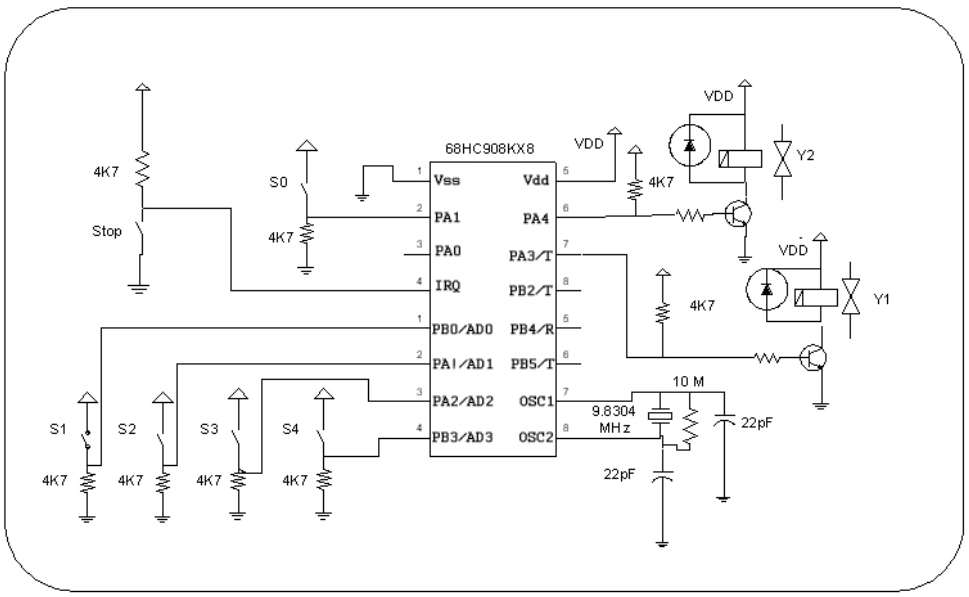
Esquema del control de un relé

Volviendo a nuestro problema

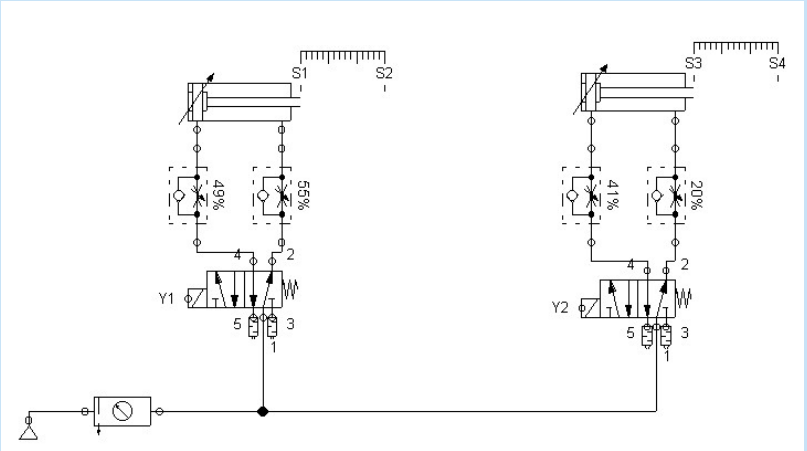
El circuito propuesto responde a una de las tantas posibilidades de conexión y control que posee el microcontrolador elegido.

Recordemos que necesitamos comandar dos cilindros neumáticos con sus correspondientes electroválvulas monoestables y sus fines de carrera eléctricos.

Circuito eléctrico:



Circuito neumático:



Y1 e Y2 son las bobinas de las electroválvulas que serán comandadas por transistores desde el microcontrolador
S1, S2, S3, y S4 son fines de carrera eléctricos para determinar la posición de los cilindros

Una vez definido el circuito eléctrico, electrónico y neumático, comenzamos a desarrollar nuestro diagrama de tareas con las distintas fases de implementación.

Veamos como sería nuestra solución:

```
PA          EQU   $0000          ;Puerto A
PB          EQU   $0001          ;Puerto B
DDRA        EQU   $0004          ;Registro de dirección de puerto A
DDRB        EQU   $0005          ;Registro de dirección de puerto B

CONFIG1     EQU   $001F          ;Registros de configuración
CONFIG2     EQU   $001E

RAM         EQU   $0040          ;comienzo de la Ram
ROM         EQU   $EC00          ;comienzo de la Rom
RESET       EQU   $FFFE          ;vector de reset

            ORG   RAM
Mem         DS    4              ;reservo memoria pero no la utilizo

            ORG   ROM
inicio
            MOV   #$01,CONFIG1 ;Inhabilito wdog
            CLR   PA           ; [1]
            CLR   PB
            MOV   #$00,DDRB    ;puerto B en entrada (fines de carrera)
            MOV   #$18,DDRA    ;puerto A bit 3 y 4 en salida (electroválvulas) bit1 en entrada
tarea:
nada:       BRCLR   1,PA,nada  ;leo estado de PA pin1 si está en «uno» comienzo la secuencia
                               ;espero que se pulse boton de marcha
            BRCLR   0,PB,nada  ;Si no están en posición el cilindro 1 y 2 no comienzo
            BRCLR   2,PB,nada
            BSET    3,PA       ;Activo electroválvula monoestable del cilindro 1 y apago la 2
            BCLR    4,PA       ; [2]
```

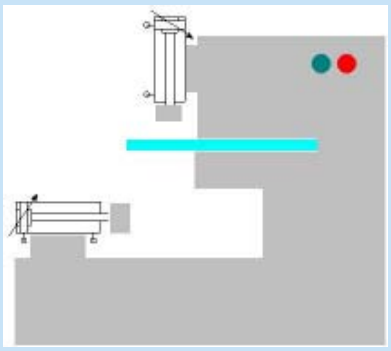
[1] *Lo invitamos a analizar la imagen a), de la secuencia que hemos ubicado al final.*

[2] *Lo invitamos a analizar la imagen b), de la secuencia que hemos ubicado al final.*

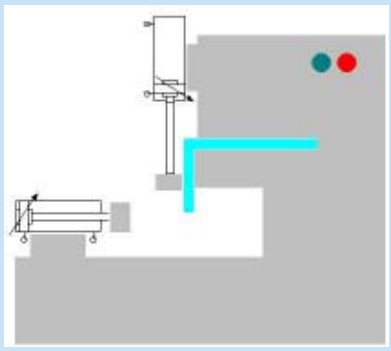
```
Sigo:     BRCLR  1,PB,Sigo     ;espero llegar al fin de carrera del cilindro 1
          BSET     4,PA        ;Activo electroválvula monoestable del cilindro 2 y mantengo la 1
                               ; [3]
espero:   BRCLR    3,PB,espero ; espero llegar al fin de carrera del cilindro 2
          BCLR   4,PA
          BCLR   3,PA          ;[4]
Nosigo:   BRCLR  2,PB,Nosigo
          BRCLR  0,PB,Nosigo   ;espero a que los dos cilindros se encuentren retraídos
          BRA    tarea

          ORG    RESET
          DW     inicio        ; Reset Vector
```

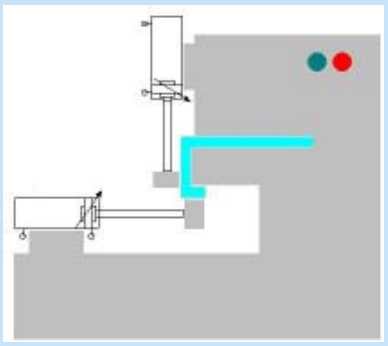
a)



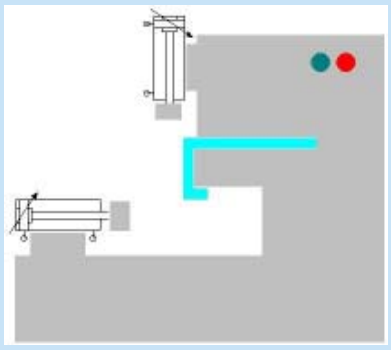
b)



c)



d)



Si queremos un pulsador de parada, podemos conectar uno en la IRQ, habilitarla y, en dicha rutina, colocamos una espera, hasta que se liberare dicho pulsador con retención.

Esto posibilita que la máquina, inmediatamente, se detenga en el procesamiento del ciclo –pero, puede ser que los cilindros aún se muevan, hasta que lleguen al final de su carrera–.

El sistema está funcionando con el sistema de clock interno (Igualmente, se indica como sería la conexión para un cristal externo de 9,8304 MHz que debería habilitarse y configurar un registro).

En este ejemplo no se desarrolla ningún sistema de debounce o antirebote de señales de entrada, de modo que se debe tener especial cuidado en caso de necesitarlo.

Veamos el código S19:

S113EC006E011F3F003F016E00056E18040300FDF6
S113EC100101FA160019000301FD18000701FD198E

[3] Lo invitamos a analizar la imagen c), de la secuencia que hemos ubicado al final.

[4] Lo invitamos a analizar la imagen d), de la secuencia que hemos ubicado al final.

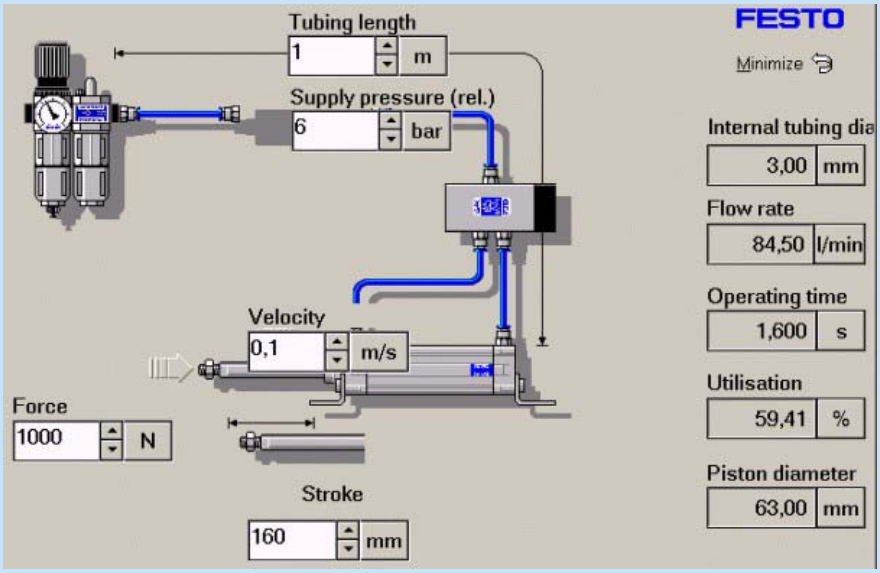
S10EEC200017000501FD0101FA20E2CD
S105FFFEEC0011
S9030000FC

Se han remarcado los códigos del programa dentro del formato S19. Dichos códigos se graban en la memoria flash, que en este microcontrolador es de, aproximadamente, 8K Bytes.

Vale la pena recordar que el programa que se graba en el microcontrolador puede ser actualizado o modificado tantas veces como sea necesario (no más de 10.000), hasta lograr el mecanismo buscado.

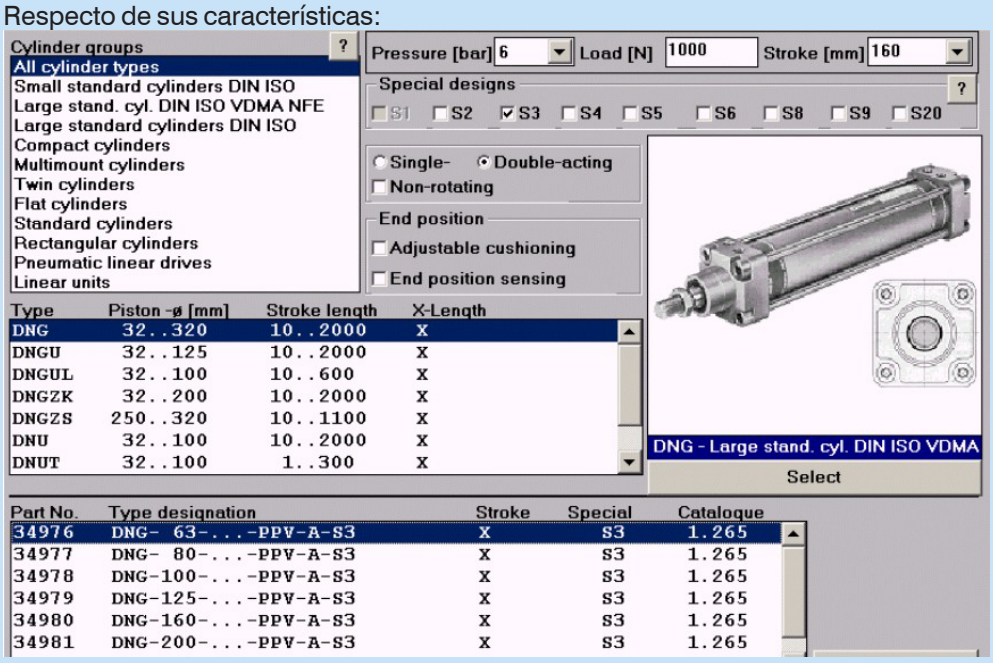
Respecto a los cilindros, hemos utilizado como referencia algunas tablas y diagramas para determinar, de acuerdo a la fuerza a desarrollar, la presión de trabajo y su carrera, el diámetro y tipo de cilindro.

Veamos los resultados:



Corresponde, entonces, utilizar cilindros de 63 mm de diámetro y carrera 160 mm a una presión de 6 bar.

Respecto de sus características:



Large stand. cyl. DIN ISO 6431, VDMA 24562, NFE 49003.1

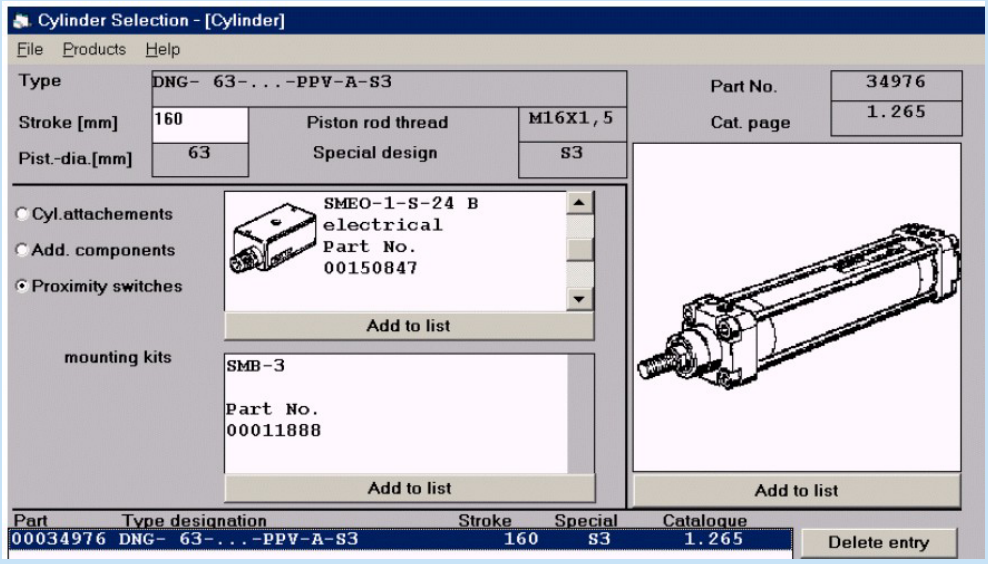
Large standard cylinders to DIN ISO 6431, VDMA 24562 and NFE 49003.1 [picture of DNG]
Cylinder mounting for any application
With adjustable end position cushioning at both ends

Profile versions:
- Type DNGU:
  Piston diameter 32-125 mm
  Stroke lengths 10-2000 mm

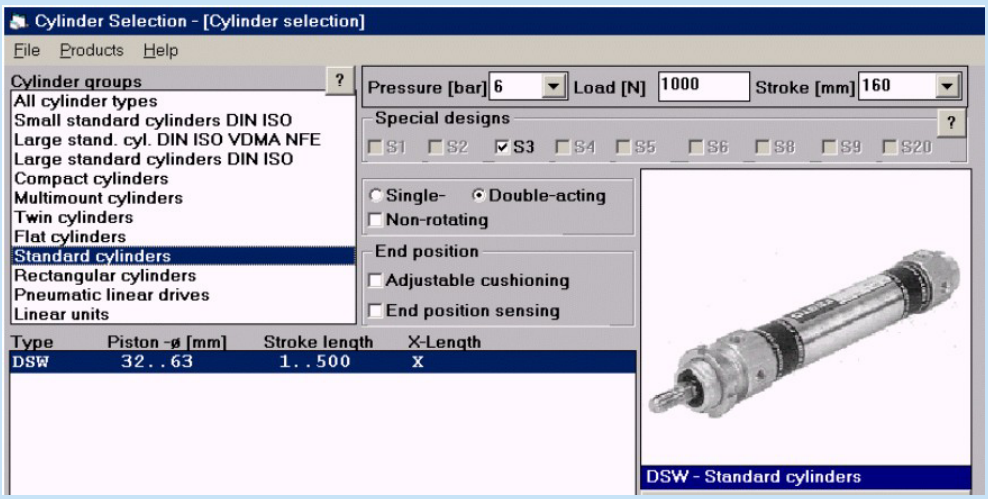
- Type DNGUL:
  With square, non-rotating piston rod
  Piston diameter 32-100 mm
  Stroke lengths 10-600 mm

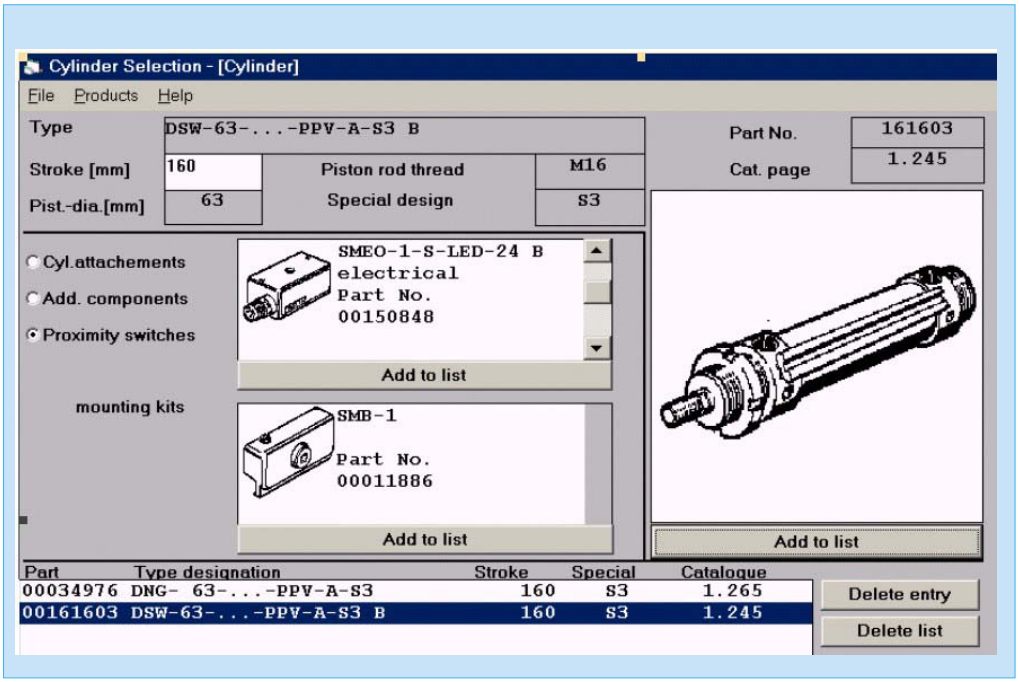
Tie rod versions:
- Type DNG, DNGZK, DNGZS
  With variable trunnion mounting [DNGZK]
  With fixed trunnion mounting [DNGZS]
  Piston diameter 32-320 mm
  Stroke lengths 10-2000 mm

- Type DNGL, DNGLZ
  With square, non-rotating piston rod
  With adjustable trunnion mounting [DNGLZ]
  Piston diameter 32-100 mm
  Stroke lengths 10-600 mm



Otra posibilidad es:




Cylinder Selection - [Cylinder]
File Products Help
Type
DSW-63-...-PPV-A-S3 B
Stroke [mm]
160
Piston rod thread
M16
Pist.-dia.[mm]
63
Special design
S3
Part No.
161603
Cat. page
1.245
Cyl.attachements
Add. components
Proximity switches
SMEO-1-S-LED-24 B
electrical
Part No.
00150848
Add to list
mounting kits
SMB-1
Part No.
00011886
Add to list
Add to list
Part
Type designation
Stroke
Special
Catalogue
00034976 DNG- 63-...-PPV-A-S3 160 S3 1.265
00161603 DSW-63-...-PPV-A-S3 B 160 S3 1.245
Delete entry
Delete list