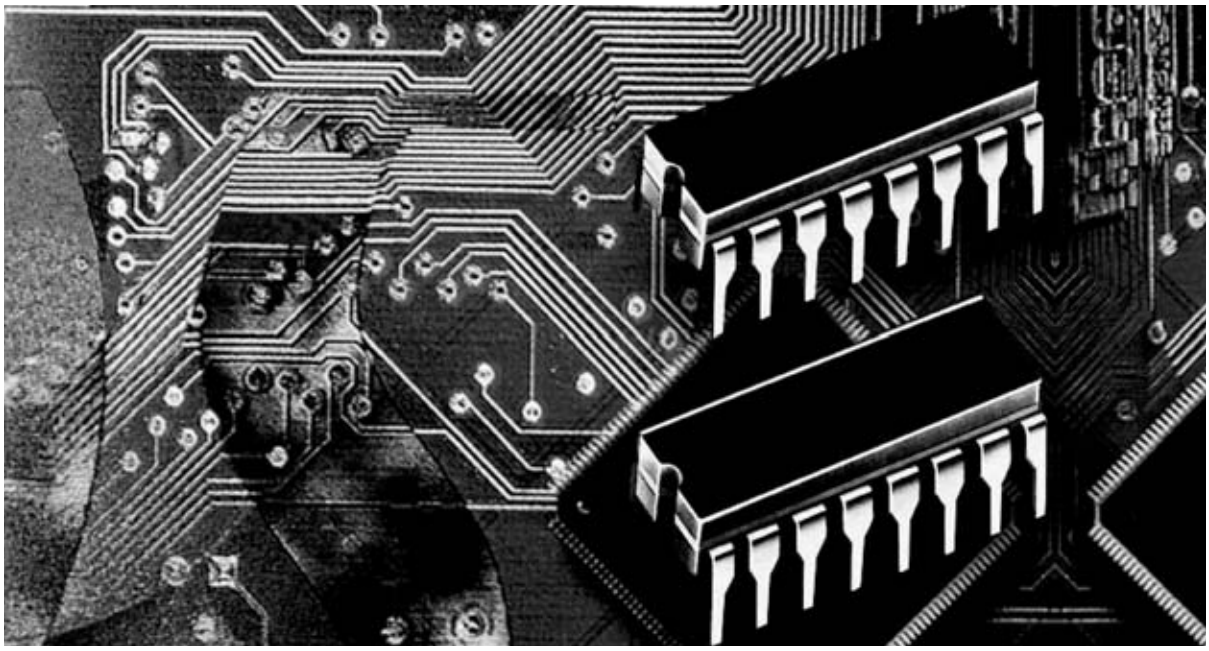

PROGRAMACIÓN CON MICROCON

En este artículo retomamos los modos de direccionamiento, cuya exposición comenzó en el tercer artículo publicado en la revista N.233. En esta ocasión abordamos el direccionamiento relativo, modo utilizado en las instrucciones de salto relativo condicional e incondicional.

El modo de direccionamiento del que nos ocupamos en este artículo afecta directamente al **valor** del **Contador de Programa** (registro **Program Counter**), es decir al registro que contiene la dirección de memoria de la instrucción siguiente a ejecutar.

Por esta razón consideramos oportuno comenzar explicando brevemente como se **ejecutan** las **instrucciones** dentro del microcontrolador y como afectan al registro **Program Counter**.

La **ejecución** de cualquier instrucción consta de varios **pasos elementales** marcados por la señal de reloj del oscilador, independientemente de si es interno o externo. Cada paso elemental se denomina **ciclo**. A pesar de que hay muchas instrucciones diferentes y constituidas por un número de **ciclos variable**, su ejecución se realiza siempre en **tres fases** comunes:

Fase de BÚSQUEDA de la instrucción (FETCH)

Tomando como base la dirección indicada por el registro Program Counter la unidad de control **adquiere** de la memoria la instrucción para su posterior análisis y ejecución. A continuación se **incrementa** el valor del registro **Program Counter** para que apunte a la dirección de memoria siguiente, donde se encuentra la próxima instrucción a ejecutar.

Fase de ANÁLISIS (decodificación) de la instrucción

Se **interpreta** el código (**op-code**) de la instrucción adquirida en la fase de búsqueda.

Fase de EJECUCIÓN de la instrucción

La unidad de control de la CPU procede a **ejecutar** la instrucción analizada y, si es

necesario, confecciona el valor del operando.

En la Fig.1 se muestra de forma gráfica el esquema de funcionamiento de las **tres fases** con cada uno de los **ciclos** que las componen.

Direccionamiento ABSOLUTO y RELATIVO

Para entender como funciona el **direccionamiento relativo**, argumento de este artículo, es importante entender lo que sucede cuando, en presencia de una instrucción de **salto**, la **CPU** sencillamente accede a una nueva zona de memoria **cambiando** el **contenido** del registro **Program Counter**, y por lo tanto la secuencia de ejecución de las instrucciones.

```
.....          ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB10h)            jp olmx
(FB13h)        ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB2Bh)    olmx    bres PORT_A,#2
.....          ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB4Ch)            jp olmx
(FB4Fh)        ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
```

El significado de la instrucción

(FB10h) **jp olmx**

es saltar a la instrucción con la etiqueta **olmx**.

El **compilador Assembler** traduce esta instrucción a un valor hexadecimal formado por

TROLADORES ST7 LITE 09 (4)

El modo más sencillo de entenderlo es **comparando** dos programas de ejemplo con secuencias diferentes de instrucciones, utilizando en el primer programa una instrucción que, por su naturaleza, **no** use el direccionamiento relativo, sustituyéndola en el segundo programa por una instrucción que **sí** utilice el direccionamiento relativo.

Comparando el comportamiento de los dos programas es muy fácil observar las diferencias y entender el funcionamiento del **direccionamiento relativo**.

Recordamos que los valores **hexadecimales** en cursiva encerrados entre paréntesis indican las **direcciones de memoria** en la que se encuentran las instrucciones. Se exponen con el único objetivo de entender el funcionamiento del direccionamiento.

Programa de EJEMPLO N.1

En este grupo de instrucciones hemos utilizado la instrucción **jp** (**jump**) de salto incondicional. El salto **siempre** se produce y es **absoluto**.

el **op-code** de la instrucción **jp**, que es **CCh** para el Assembler del micro ST7, y por la **dirección de memoria** del operando **olmx**. Ya que la etiqueta **olmx**, como se puede observar en el listado, está asociada a la dirección **FB2Bh**, después de la compilación el código operativo de esta instrucción está formado por **3 bytes**: **CCFB2Bh**.

Cuando se ejecuta el programa, para **saltar** a la instrucción con etiqueta **olmx** la **CPU** introduce en el registro **Program Counter** el valor del operando (**FB2Bh**).

En la práctica, de los tres bytes de **CCFB2Bh** que componen el código de la instrucción, los dos de la derecha (**FB2Bh**) corresponden la dirección de memoria a la que se salta cuando se ejecuta la instrucción. En este caso se trata de una dirección **absoluta**, ya que **todas** las **instrucciones jp** de **salto** con el operando **olmx** incluidas en el programa, como por ejemplo:

(FB4Ch) **jp olmx**

generan un código que siempre tiene el mismo valor: **CCFB2Bh**.

Por lo tanto, una vez lanzado el programa, cuando se ejecuta la instrucción **jp olmx**, el registro **Program Counter** siempre se modifica con la dirección presente en los dos bytes de la derecha, en nuestro caso **FB2Bh**. De esta forma el programa salta desde cualquier punto de la memoria a la instrucción con etiqueta **olmx**, independientemente de que la instrucción **jp** se encuentre **antes** o **después** de la dirección de la etiqueta **olmx**. Es importante recordar que **Program Counter** es un registro de **16 bits** (**2 bytes**), por tanto también en este ejemplo se modifican los dos bytes.

Programa de EJEMPLO N.2

En este grupo de instrucciones hemos utilizado una instrucción de salto incondicional, pero con direccionamiento **relativo**: **jra** (**jump relative always**).

En particular hemos reemplazado las instrucciones

(FB10h) **jp olmx**
(FB4Ch) **jp olmx**

por las instrucciones:

(FB10h) **jra olmx**
(FB4Ch) **jra olmx**

```
.....          ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB10h)             jra olmx
(FB12h)        ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB2Bh)   olmx      bres PORT_A,#2
.....          ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
(FB4Ch)             jra olmx
(FB4Eh)        ..... ; instrucciones del programa
.....          ..... ; instrucciones del programa
```

Hemos introducido intencionadamente **dos** instrucciones idénticas **jra olmx**, si bien se encuentran en diferentes direcciones de memoria, una **antes** de la dirección asociada a la etiqueta **olmx** y otra **después**.

Comencemos analizando la primera, es decir:
(FB10h) **jra olmx**

Cuando el compilador traduce esta instrucción reemplaza la instrucción **jra** por su **op-code** (**20h** para el Assembler del micro ST7), y como dirección de salto (**olmx**) no inserta la dirección de memoria del operando sino la **diferencia** entre esta dirección (**FB2Bh**) con la dirección de memoria de la **instrucción siguiente** a la de salto (**FB12h**), es decir:

FB2Bh - FB12h = 19h

Este valor se denomina **desplazamiento**.

Una vez realizada la compilación el código de la instrucción no es de tres bytes, como en el ejemplo anterior, sino de **2 bytes**: **2019h**. Resumiendo, se puede decir que el **direccionamiento relativo** es la **distancia** (en **bytes**) entre la instrucción siguiente a la de salto (**jra**) y la etiqueta (**olmx**) a la que se salta. Cuando se ejecuta la instrucción **2019h (jra olmx)** la CPU **incrementa** el registro **Program Counter**, que contiene ya la dirección de memoria de la instrucción siguiente a ejecutar (en nuestro caso **FB12h**), con el valor del desplazamiento (**19h**). Así el programa salta a **olmx**, en efecto: **FB12h + 19h = FB2Bh**.

La primera consecuencia que se puede observar es que mientras en el primer ejemplo el operando **olmx** fue traducido a un código de 2 bytes, con la instrucción **jra** el operando **olmx** ocupa solo **1 byte**. Esto permite ahorrar espacio, aunque la distancia máxima que se puede tener entre una instrucción de salto **relativo** y la correspondiente **etiqueta** es de **256 bytes** (de 0 a 255), mientras que con la instrucción **jp** de salto absoluto no se fijan límites en la distancia entre las instrucciones.

En realidad, ya que tiene que ser posible saltar a una etiqueta que se encuentre definida tanto **antes** como **después** de la instrucción de salto relativo, la **distancia máxima** permitida es de **-127** a **+128 bytes** de la instrucción siguiente a la de salto. Llegados a este punto nos encontramos un primer obstáculo ¿Cómo es posible indicar un valor de **desplazamiento negativo** ya que, en principio, el binario **no** permite la posibilidad de representar el signo menos?. La realidad es que sí es posible ya que existente un sistema de numeración

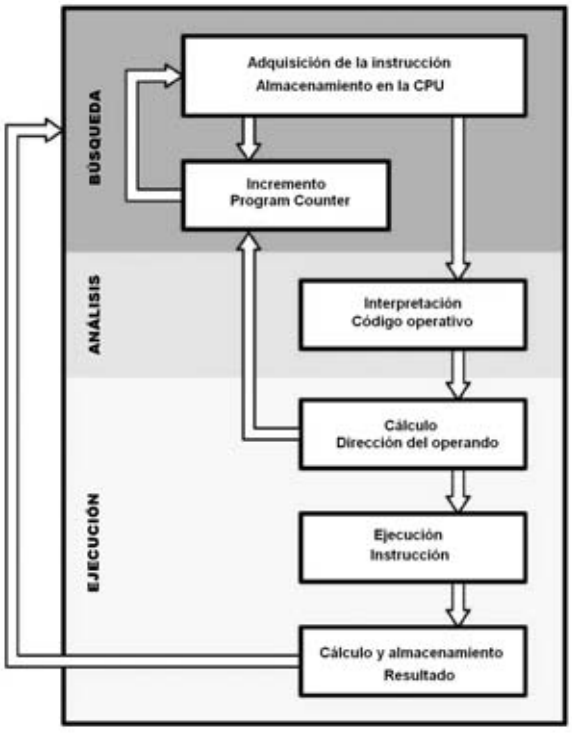

Fig.1 Esquema de funcionamiento de la ejecución de una instrucción. La unidad de control activa una secuencia de pasos al ritmo marcado por la señal de reloj. Como se puede observar el proceso se divide en tres fases: Búsqueda, análisis y ejecución propiamente dicha. En cuanto se adquiere la instrucción la Unidad de Control incrementa el valor del registro Program Counter, pudiendo ser posteriormente incrementado por un operando.

binaria para números negativos: **El complemento a 2**.

Para explicar como se realiza este proceso hemos introducido la segunda instrucción **jra**, es decir la definida después de la etiqueta **olmx**:

(FB4Ch) **jra olmx**

En esta instrucción la **dirección de memoria** es **más alta** que la dirección de memoria de la etiqueta **olmx**.

Como hemos visto anteriormente cuando el **compilador** traduce esta instrucción reemplaza la instrucción **jra** por su **op-code** (**20h**) y como dirección de salto inserta la **diferencia** entre la dirección de memoria de la instrucción con etiqueta **olmx** (**FB2Bh**) y la dirección de memoria de la instrucción siguiente a la de salto (**FB4Eh**), es decir:

FB2Bh - FB4Eh = FFDDh

Ya que el primer término de la sustracción (**FB2Bh**) es más **pequeño** que el segundo término (**FB4Eh**) el resultado es negativo. Para expresar este resultado se utiliza el **complemento a 2**.

NOTA: En artículos anteriores se expuso el concepto de **valor negativo** al abordar el **Flag N** del registro **Condition Code**. Recordamos que el microcontrolador interpreta como **positivos** los valores comprendidos entre **0h** y **7Fh** (**0** y **127** en decimal) y como **negativos** los valores comprendidos entre **80h** y **FFh** (**128** a **255** en decimal).

Resumiendo, el compilador utiliza como valor de **desplazamiento** el byte menos significativo (**LSB**) de la sustracción, es decir **DDh**, y traduce la instrucción entera a un código de instrucción de **2 bytes**: **20DDh**.

Cuando la ejecución del programa llega a la instrucción de salto **jra olmx**, la CPU utiliza la **ALU** (Unidad Aritmético-Lógica) para ejecutar el cálculo. Sin introducirnos demasiado en los detalles internos que no aclaran en nada el mecanismo de direccionamiento, se realiza la sustracción en **complemento a 2**:

FB2Bh - FB4Eh = FFDDh

Si queremos **comprobar** la operación solo hay que sumar **FFDDh** a **FB4Eh**, el resultado debería ser **FB2Bh**, pero en realidad es **01FB2Bh**, es decir ocupa **3 bytes**.

Este suceso se denomina **desbordamiento** (**overflow**) ya que el resultado ocupa **más espacio** que el espacio donde se tiene que almacenar, en este caso los **2 bytes** del registro **Program Counter**. El valor es truncado a **2 bytes**, por lo que el registro **Program Counter** pasa a valer **FB2Bh** que es la dirección de memoria de la etiqueta **olmx**. De esta forma el programa salta hacia atrás. La explicación indudablemente es larga. Ahora bien, está

justificada por el hecho de que el modo de direccionamiento **relativo** es muy importante.

En efecto, además de la instrucción **jra** el resto de **instrucciones de salto condicional** (ver Tabla N.1) utilizan también el direccionamiento relativo. Son muy utilizadas en los programas. A medida que se adquiere práctica con la programación el empleo del **direccionamiento relativo** se hace muy sencillo y permite escribir bloques y subrutinas que se pueden emplear más de una vez en el mismo programa, o incluso utilizarse en programas diferentes.

Siempre que se utilice una instrucción con direccionamiento relativo hay que tener presente la **distancia** a la **etiqueta** para que no se encuentre **fuera del rango de direccionamiento**. Si esto sucede el compilador muestra un mensaje de **error** como el reproducido en la Fig.2.

Cuando expliquemos la utilización de cada una de las instrucciones os enseñaremos una técnica para **no** tener problemas casi nunca. Ahora vamos a exponer los dos modos de direccionamiento relativo.

DIRECCIONAMIENTO RELATIVO

Como hemos visto, el **direccionamiento relativo** implica esencialmente al registro **Program Counter** haciéndolo variar en un valor que oscila entre **-127** y **+128** bytes. Este direccionamiento tiene **dos modos**:

Relativo Directo
Relativo Indirecto

Relativo DIRECTO

En el ejemplo anterior hemos utilizado la instrucción **jra olmx** (ver listado del Programa de ejemplo N.2). Esta instrucción utiliza el modo **relativo directo**.

En este modo el **valor** del **operando** (**desplazamiento**) constituye la distancia entre la instrucción y la etiqueta, almacenándose en el operando de la instrucción. En este caso el valor del registro **Program Counter** es incrementado o decrementado directamente con valor del **desplazamiento**.

Si se comete un error el **compilador** indica si se ha producido un desplazamiento más grande del permitido.

Relativo INDIRECTO

En este modo el **desplazamiento** efectivo no se almacena como operando de la instrucción sino que es **memorizado** en un byte de memoria **RAM** al que se asociada normalmente una etiqueta.

Por tanto el operando de la instrucción no contiene el valor del desplazamiento sino la **dirección de memoria** dónde está contenido este valor.

Para distinguir esta modo del directo el operando se encierra entre **corchetes**.

Para aclarar el uso de este modo a continuación proponemos un listado de ejemplo que también utiliza la instrucción **jra** pero con direccionamiento **relativo indirecto**.

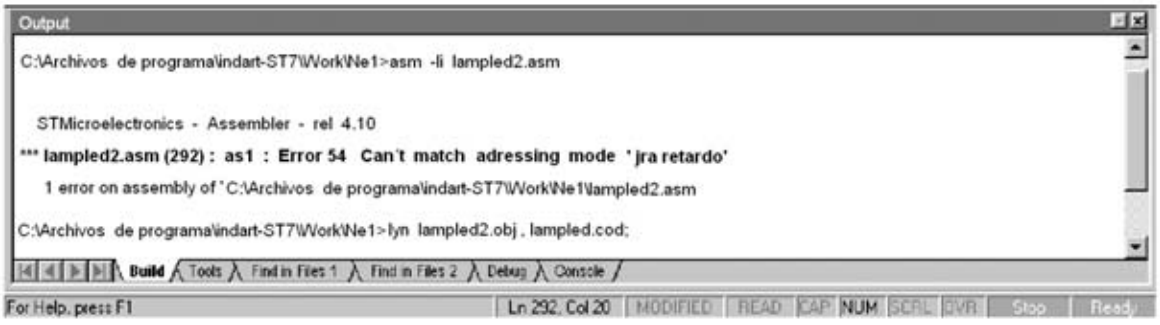

Fig.2 En esta figura se reproduce el tipo de error señalado por el compilador Assembler cuando el valor del operando de la instrucción de salto relativo no respeta el rango de -127 a +128 (ver Error 54: Can't match Addressing mode).

```
(0087h)   TEST      DS.B 1
.....     ..... ; instrucciones del programa
.....     ..... ; instrucciones del programa
(FA33h)           ld a,#0Ch
(FA35h)           ld TEST,a
(FA37h)           jra [TEST]
(FA3Ah)           bset PORT_A,#0
.....     ..... ; instrucciones del programa
.....     ..... ; instrucciones del programa
(FA46h)           bres PORT_A,#0
```

Con la primera instrucción:

(0087h) **TEST DS.B 1**

se define en el área **Data Memory** una variable de 1 byte de longitud llamada **TEST** y localizada en la dirección **87h**.

Cuando el compilador encuentra la instrucción:

(FA33h) **ld a,#0Ch**

en el acumulador **a** se carga el valor **0Ch**, mientras que la instrucción:

(FA35h) **ld TEST,a**

lleva el contenido de **a** (**acumulador**) a la variable **TEST**. Después de esta instrucción la variable **TEST** contiene el valor **0Ch**.

Cuando el compilador encuentra la instrucción:

(FA37h) **jra [TEST]**

traduce **jra** en el op-code **20h**.

Como el operando **TEST** está encerrado entre **corchetes** se reconoce el modo **indirecto** y se antepone al op-code **20h** el valor **92h**. Por último se agrega la dirección de **TEST** (**87h**). El resultado final de la compilación de esta instrucción es **922087h**.

Cuando al ejecutar el programa se procesa la instrucción **922087h**, el registro **Program Counter**, que contiene ya la dirección de memoria de la instrucción siguiente a ejecutar (**FA3Ah** en nuestro caso), se **incrementa** con el **valor contenido** en la variable **TEST** (dirección **87h**), es decir con **0Ch**.

El programa salta pues a FA3Ah + 0Ch, es decir a:

(FA46h) **bres PORT_A,#0**

instrucción que **pone a 0** el terminal **0** del **Puerto A**.

Llegado este punto parece bastante evidente la **diferencia** entre los dos modos de direccionamiento relativo.

En modo **Directo** el operando indicado en la instrucción de salto es la dirección de una etiqueta, por tanto el Compilador Assembler realiza el cálculo de la **distancia** que separa la instrucción de salto de la etiqueta y controla que esté comprendida entre **-127** y **+128**. La instrucción en formato ejecutable contendrá el valor de esta distancia y no se puede modificar. Es decir, la instrucción de direccionamiento directo **jra olmx** siempre salta únicamente a la etiqueta **olmx**.

En modo **indirecto** el operando que indicamos en la instrucción de salto es la **dirección** de una **variable** en memoria RAM que contiene el valor de incremento/decremento del registro Program Counter. En este modo de direccionamiento el compilador se limita a controlar que la variable esté definida en el área **Program Space**. No se producen errores de salto ya que al tener la variable un byte de longitud la posibilidad de salto está dentro de los límites de **-127** a **+128**.

Sin embargo, en este caso el **salto** puede ser **variable** y por tanto la misma instrucción puede hacer saltar el programa a puntos diferentes. En otras palabras, la instrucción **jra [TEST]** hace saltar al programa a las direcciones de memoria según el valor presente en **TEST**. El único problema que puede plantearse es el cálculo del número a introducir en la variable **TEST** para obtener el salto a la dirección deseada.

Una ayuda bastante importante es la proporcionada por algunas **directivas Assembler** que trataremos en el momento adecuado. Para ahora nos detenemos aquí con el tema del direccionamiento.

RESUMEN
DIRECCIONAMIENTO RELATIVO

Las instrucciones que utilizan el modo de **direccionamiento relativo** son las instrucciones de **salto relativo condicional**, como por ejemplo **jrt jrf jrm** etc. (ver Tabla N.1), la instrucción **jra** de **salto relativo incondicional** y la instrucción **callr**. Como la instrucción **call**, la instrucción **callr** almacena en la **Pila** (**Stack Memory**) la dirección de la instrucción siguiente a la solicitud de inicio de la subrutina, pero en este caso la **dirección** es el resultado de **sumar** el valor del **desplazamiento** y el valor contenido en el registro **Program Counter** de la instrucción siguiente.

NOTA: El funcionamiento y empleo del **Puntero de Pila** (registro **Stack Pointer**) ya ha sido detallado en artículos anteriores de esta serie.

El modo **relativo** se utiliza para modificar el valor contenido en el registro Program Counter. En efecto, el operando de la instrucción añade un valor de **desplazamiento** (entre **-127** y **+128**) al valor del registro Program Counter. En el caso de que la instrucción trabaje en modo **relativo directo** el valor del registro **Program Counter** se incrementa o decrementa con el **valor del desplazamiento**, mientras que si trabaja en modo **relativo indirecto** el operando (entre corchetes) contiene la **dirección del desplazamiento**.

EJEMPLO de OP-CODE

A continuación exponemos algunos ejemplos de instrucciones con modo de **direccionamiento relativo** en formato **Assembler** y en formato **ejecutable** (ver columna **op-code**). Las abreviaturas utilizadas empleadas son las que se utilizan en los **manuales** de las instrucciones Assembler de los micros ST7. Su significado es el siguiente:
rel = relativo directo
[rel] = relativo indirecto

Modo	Instrucción Assembler	Op-code
rel	jrne loop	26 XX
[rel]	jrne [10h]	92 26 XX

26: Valor del código de operación de la instrucción **jrne**. Si, por ejemplo, hubiéramos utilizado la instrucción **jrpl** el valor habría sido **2A**.

TABLA N.1 INSTRUCCIONES Y DIRECCIONAMIENTO

Mnemo Instrucción	Descripción Instrucción	Direccionamiento rel	Direccionamiento [rel]
ADC	Addition with Carry		
ADD	Addition		
AND	Logical And		
BCP	Logical Bit compare		
BRES	Bit reset		
BSET	Bit set		
BTJF	Bit test and Jump if false		
BTJT	Bit test and Jump if true		
CALL	Call subroutine		
CALLR	Call subroutine relative	•	•
CLR	Clear		
CP	Compare		
CPL	One Complement		
DEC	Decrement		
HALT	Halt		
INC	Increment		
IRET	Interrupt routine return		
JP	Absolute Jump		
JRA	Jump relative always	•	•
JRT	Jump relative	•	•
JRF	Never Jump	•	•
JRIH	Jump if Port INT pin = 1	•	•
JRIL	Jump if Port INT pin = 0	•	•
JRH	Jump if H = 1	•	•
JRNH	Jump if H = 0	•	•
JRM	Jump if I = 1	•	•
JRNM	Jump if I = 0	•	•
JRMI	Jump if N = 1 (minus)	•	•
JRPL	Jump if N = 0 (plus)	•	•
JREQ	Jump if Z = 1 (equal)	•	•
JRNE	Jump if Z = 0 (not equal)	•	•
JRC	Jump if C = 1	•	•
JRNC	Jump if C = 0	•	•
JRULT	Jump if C = 1	•	•
JRUGE	Jump if C = 0	•	•
JRUGT	Jump if (C + Z = 0)	•	•
JRULE	Jump if (C + Z = 1)	•	•
LD	Load		
MUL	Multiply		
NEG	Negate (2's complement)		
NOP	No operation		
OR	Or operation		
POP	Pop from the Stack		
POP	Pop CC		
PUSH	Push onto the Stack		
RCF	Reset carry flag		
RET	Subroutine return		
RIM	Enable Interrupts		
RLC	Rotate left true C		
RRC	Rotate right true C		
RSP	Reset stack pointer		
SBC	Subtract with Carry		
SCF	Set carry flag		
SIM	Disable interrupts		
SLA	Shift left Arithmetic		
SLL	Shift left Logic		
SRA	Shift right Arithmetic		
SRL	Shift right Logic		
SUB	Substraction		
SWAP	Swap nibbles		
TNZ	Test for Neg & Zero		
TRAP	S/W trap		
WFI	Wait for interrupt		
XOR	Exclusive OR		

92: Valor que identifica el modo **indirecto**. Siempre es el mismo para cualquier instrucción de **salto relativo indirecto**.
XX: Valor del operando. Contiene el valor del desplazamiento.