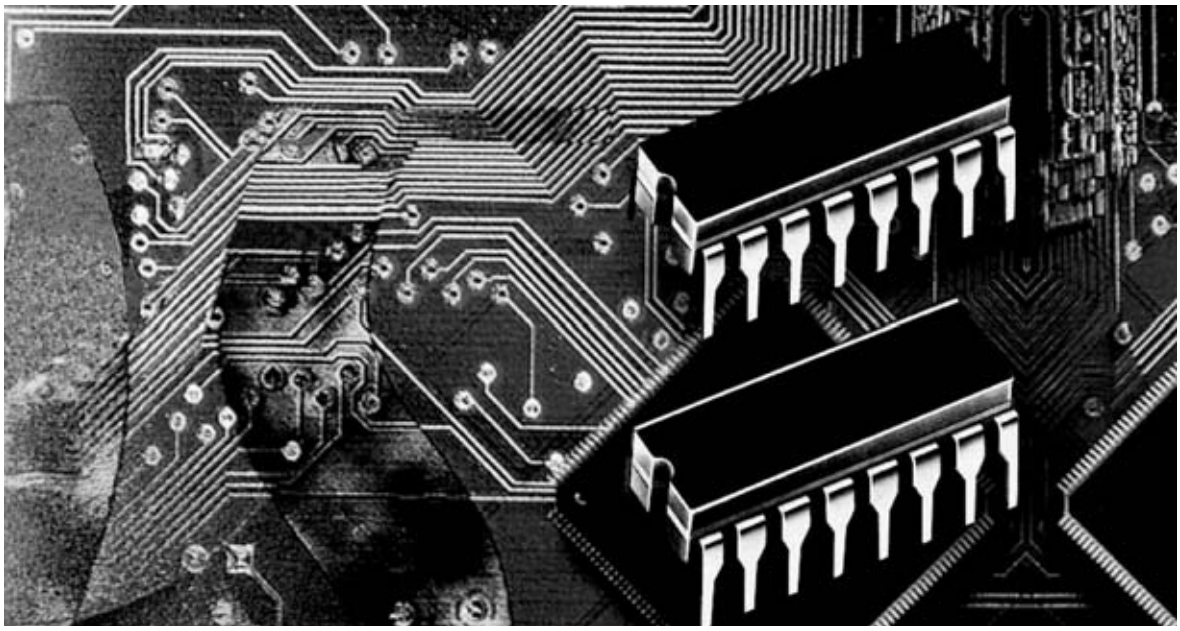

Programación con microcon

Una de las peculiaridades del lenguaje Assembler para los micros ST7 es la posibilidad de direccionar los operandos de una instrucción con varios modos diferentes. Hoy vamos a aprender a reconocer y a utilizar los modos de direccionamiento indirectos y los modos indexados indirectos.

Algunos lectores, después de haber leído los artículos anteriores sobre direccionamiento, nos han escrito preguntándonos cual es el **modo de direccionamiento** mejor. A esta pregunta sólo podemos contestar que no existe un modo mejor que los otros, ya que de ser así solo existiría ese modo de direccionamiento. Existen **diferentes modos** ya que cada uno es el más adecuado en función de la estructura del programa.

Por esta razón se hace necesario conocer todos los **modos de direccionamiento** y, sobre todo, comprender su **funcionamiento**. Los ejemplos que os proponemos son muy sencillos y están realizados para alcanzar este objetivo.

Cuando nos ocupemos de cada una de las **instrucciones Assembler**, haremos referencia a los modos de direccionamiento ya analizados. Una vez leídos estos artículos ya no tendréis dudas sobre la elección de los modos a utilizar en vuestros programas.

DIRECCIONAMIENTO INDIRECTO

Para explicar la lógica y las características del direccionamiento **indirecto** vamos a hacer un desarrollo similar al del direccionamiento **directo**, presentado en la revista **N.233**.

Recordando el modo de direccionamiento **directo**, el operando define el **valor contenido** en la dirección de memoria, con instrucciones del tipo:
ld a,pippo

El valor contenido en la dirección de memoria en la que está definida la variable **pippo** se obtiene directamente y se carga en el acumulador **A**. En modo Indirecto una instrucción de este tipo se escribe de la siguiente forma:

ld a,[pippo]

En el acumulador **A** no se carga el valor contenido a la dirección de memoria **pippo** sino que, en este caso, se carga en el acumulador el **contenido** de la **posición de memoria** cuya dirección se encuentra en la dirección de memoria **definida** por la variable **pippo**. Por esta razón se llama direccionamiento indirecto: La variable **no** contiene el **operando** sino la **dirección** que contiene el **operando**.

en el variable **pippo** sino el valor contenido en la **dirección** a la que **apunta** el contenido del variable **pippo**, como se muestra gráficamente en la Fig.2.

Después de la ejecución de la instrucción el registro **A** contiene el valor **18h** (ver Fig.2). La instrucción con modo **directo**, es decir con el operando **sin corchetes**, habría cargado en **A** directamente el valor **9Ch**.

La instrucción **ld a,[pippo]** en formato ejecutable tiene un **código de operación** (**op-code**) **92B67B**, donde **92** indica que se trata de una instrucción de direccionamiento **indirecto**, **B6** es el código propio de la instrucción **ld a,[dirección]** y **7B** es la dirección de la variable con etiqueta **pippo**.

troladores ST7 LITE 09 (5)

En principio puede parecer un poco difícil de entender, vamos a aclararlo más. En la Fig.1 hemos detallado las partes que componen una instrucción Assembler en código fuente.

Supongamos que la variable **pippo** ha sido definida en la dirección **007Bh** de memoria RAM. La instrucción:

ld a,[pippo]

trabaja con direccionamiento **indirecto**. En el acumulador **A no** se carga el valor contenido

Dado que el código de operación tiene **tres bytes** (**92B67B**), la dirección siguiente del Contador de Programa (Program Counter) corresponde a la dirección de la instrucción + **3 bytes**. Por tanto, si la instrucción estuviera en **043Dh**, el Contador de Programa pasaría a **043Dh + 3 = 0440h**.

La primera característica a tener presente en el direccionamiento **indirecto** es que también permite acceder de forma **indirecta** a la dirección de memoria del **operando destino**.

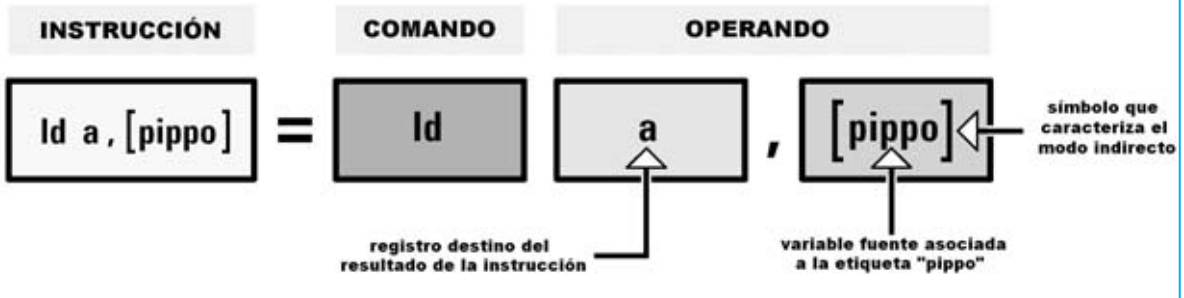

Fig.1 Cuando el operando de una instrucción, ya sea fuente o destino del resultado de la instrucción, está encerrado entre corchetes [ ], estamos en presencia de una instrucción con direccionamiento indirecto.

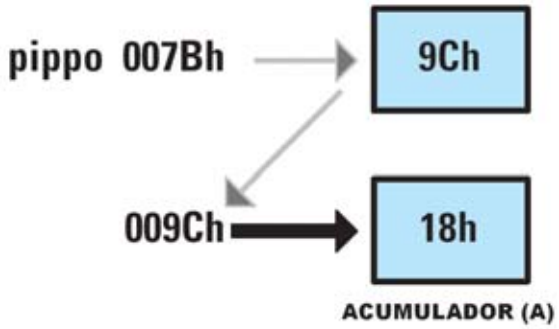

Fig.2 Si la variable PIPPO está definida en la dirección 007Bh y contiene el valor 9Ch, después de la instrucción "ld a,[pippo]" en el acumulador A se carga el valor contenido en la dirección 9Ch, es decir 18h.

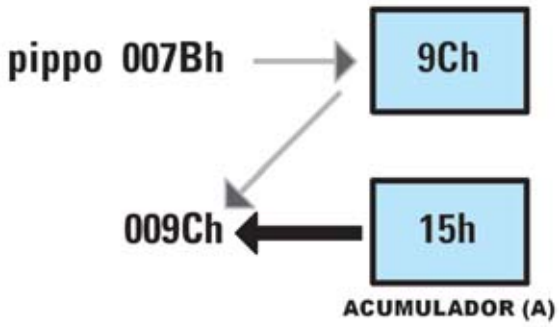

Fig.3 Si la variable PIPPO está definida en la dirección 007Bh y contiene el valor 9Ch, después de la instrucción "ld [pippo],a" el valor 15h contenido en el registro A se carga en la dirección 9Ch.

En otras palabras, también existe la instrucción:

ld [pippo],a

Obviamente en este caso el contenido del registro **A no** se carga en la dirección en la que está definida la variable **pippo** sino en la dirección a la que **apunta** el contenido de la variable **pippo**.

Suponiendo que el registro **A** contiene el valor **15h** y **pippo** está definida, como ya hemos mencionado, en la dirección **007Bh** (que contiene el valor **9Ch**), esta instrucción carga el valor **15h** en la dirección **9Ch** (ver Fig.3).

La instrucción con modo **directo**, es decir con el operando **sin corchetes**, habría cargado directamente en pippo (dirección **007Bh**) el valor **15h**.

También la instrucción **ld [pippo],a** tiene un código de operación de tres bytes, en este caso **92B77B**, donde **92** indica que se trata de una instrucción de direccionamiento **indirecto**, **B7** es el código propio de la instrucción **ld [dirección],a** y **7B** es la dirección de la variable con etiqueta **pippo**.

Ya que el código de operación tiene **tres bytes** (**92B77B**), la dirección siguiente del Contador de Programa corresponde a la dirección de la instrucción + **3 bytes**. Por tanto, si la instrucción estuviera en **043Dh**, el Contador de Programa pasaría a **043Dh + 3 = 0440h**.

Hay que señalar llegado este punto que, al igual que el modo directo, también el modo de direccionamiento indirecto puede **indexarse**, utilizando uno de los registros índice **X -Y**, para acceder **indirectamente** a una **posición de memoria**.

Por tanto, el **direccionamiento indirecto** tiene cuatro diferentes modos operativos que a continuación pasamos a analizar pormenorizadamente:

- Indirecto Corto (Short)
- Indexado Indirecto Corto (Short)
- Indirecto Largo (Long)
- Indexado Indirecto Largo (Long)

INDIRECTO CORTO (SHORT)

Una vez aclarado el **concepto** de direccionamiento **indirecto**, vamos a presentar un ejemplo que ayude a entender mejor esta forma de direccionamiento.

En primer lugar definimos en **Data Memory** una serie de variables:

(0087h)	**VALAD1**	**DS.B 1**
.....	.....	.....
(008Bh)	**VALAD2**	**DS.B 1**
.....	.....	.....
(0094h)	**PUNTAT**	**DS.B 1**

NOTA: Los valores **hexadecimales** entre paréntesis situados a la izquierda representan **direcciones** de **memoria**. Se trata de valores **hipotéticos** expuestos con fines didácticos. Las tres direcciones son intencionalmente no contiguas.

Las siguientes instrucciones se alojan en **Program Memory**:

```
(FA13h)            ld      a,#VALAD2
(FA15h)            ld      PUNTAT,a
(FA17h)            call    caricad
.............  ..............  ..............  ..............
(FAD0h)            ld      a,#VALAD1
(FAD2h)            ld      PUNTAT,a
(FAD4h)            call    caricad
.............  ..............  ..............  ..............
(FAE2h)        caricad
.............  ..............  ..............  ..............
(FAE9h)            btjf    ADCCSR,#7,$
(FAFCh)            ld      a,ADCDR
(FAFFh)            ld      [PUNTAT],a
(FB03h)            ret
```

Empezamos analizando el programa con la primera instrucción:

(FA13h) **ld** **a,#VALAD2**

Esta instrucción carga en el acumulador **A** la dirección de la variable **VALAD2**. En efecto, el símbolo corchete "**#**" que precede al operando indica que se trata de una instrucción con direccionamiento **inmediato** (ver revista **N.233**).

Al compilar esta instrucción se traduce a formato ejecutable con un código de operación **A68B**, donde **A6** es la traducción en formato ejecutable de **ld a,#valor**, mientras que **8B** es el valor **inmediato** del operando (que corresponde a la dirección de memoria de la variable **VALAD2**)

Cuando se ejecuta la instrucción el micro carga el valor **8Bh** en el acumulador **A**.

Con la instrucción:

(FA15h) **ld** **PUNTAT,a**

de direccionamiento directo, se carga en la variable **PUNTAT** el valor contenido en el acumulador **A**.

Al compilar esta instrucción se traduce a formato ejecutable con un código de operación **B794**, donde **B7** es la traducción en formato ejecutable de **ld variable,a**, mientras que **94** es la dirección de memoria de la variable **PUNTAT**.

Cuando se ejecuta esta instrucción el micro carga en la variable **PUNTAT** el contenido del acumulador **A**, es decir **8Bh** (que corresponde a la dirección de memoria de la variable **VALAD2**).

La instrucción siguiente, es decir:

(FA17h) **call** **caricad**

lanza la ejecución de la subrutina **caricad**. Por tanto el programa salta a la etiqueta **caricad**:

(FAE2h) **caricad**

y almacena en la **pila** (**Stack Memory**) la dirección de retorno.

NOTA: El funcionamiento de la pila y de las subrutinas ha sido explicado en artículos anteriores.

La instrucción de la subrutina:

(FAE9h) **btjf** **ADCCSR,#7,$**

realiza una lectura A/D y almacena el resultado en una variable.

El funcionamiento del **Conversor A/D** se tratará en próximos artículos. De momento adelantamos que el registro **ADCCSR** controla la conversión A/D y que la instrucción **btjf ADCCSR,#7,$** literalmente significa "salta a **$** si el bit **7** del registro de control **ADCCSR** es **0**".

NOTA: la instrucción **btjf** es el acrónimo de **B**it **T**est **J**ump if **F**alse, es decir controla el bit y salta si 0.

El terminal **7** del registro **ADCCSR** se pone a **0** cuando se activa una conversión **A/D**

(Analógico/Digital) y queda así hasta que se realiza la conversión. Una vez realizada la conversión este terminal pasa a **1** y el dato digital convertido se almacena en el registro **ADCDR**.

El símbolo **$** representa al "Contador de Programa actual", que en nuestro ejemplo vale **FAE9h**. Por tanto, la instrucción **btjf ADCCSR,#7,$** realiza saltos sobre sí misma hasta que se realiza la conversión A/D.

Suponiendo que el resultado de la conversión A/D es **1Fh**, el registro **ADCDR** contendrá este valor.

Ahora, como hemos expuesto, vamos a almacenar este resultado en una variable, comenzando por moverlo al acumulador **A** con la siguiente instrucción:

```
(FAFCh)        ld      a,ADCDR
```

y, a continuación:

```
(FAFFh)        ld      [PUNTAT],a
```

A primera vista puede parecer que el valor que se encuentra en el acumulador **A** se almacena en el variable **PUNTAT**, no es así ya que los corchetes indican la utilización de un **direccionamiento indirecto corto**.

Al compilar esta instrucción se traduce a formato ejecutable con un código de operación de tres bytes: **92B794**, donde **92** indica que se trata de una instrucción de direccionamiento indirecto, **B7** es la traducción en formato ejecutable de **ld variable,a**, mientras que **94** es la dirección de memoria de la variable **PUNTAT**.

Cuando se ejecuta la instrucción, el valor presente en el acumulador **A**, es decir **1Fh**, se carga en la dirección de memoria indicada por el valor presente en la variable **PUNTAT**.

Puesto que en **PUNTAT** cargamos la dirección de la variable **VALAD2**, es decir **8Bh**, el valor **1Fh** se almacena en **VALAD2** (dirección **8Bh**). Con este modo de direccionamiento la variable **PUNTAT** se ha utilizado como **puntero de memoria**.

Con la instrucción:

```
(FB03h)        ret
```

finaliza la subrutina provocando el **retorno** al programa principal.

El programa continúa con las siguientes instrucciones:

```
(FAD0h)        ld      a,#VALAD1
(FAD2h)        ld      PUNTAT,a
(FAD4h)        call    caricad
```

Llegados a este punto el valor resultante de la conversión A/D se almacena, mediante estas instrucciones, en la variable **VALAD1**.

Con este ejemplo hemos querido mostrar como utilizando el modo de direccionamiento **indirecto** se pueden utilizar subrutinas con parámetros sin modificarla. En efecto, solo hemos modificado la dirección de la variable, mientras que la subrutina **caricad** ha quedado inalterada.

El modo **indirecto** se define como **short** (**corto**) cuando el direccionamiento se realiza con una **variable** de **un byte** de longitud, lo que implica un rango de **00h** a **FFh**.

INDEXADO INDIRECTO CORTO (SHORT)

El modo **indexado indirecto corto** es un modo de direccionamiento análogo al anterior pero que utiliza el registro índice **X** o el **Y**, de ahí el término "**indexado**".

Merece la pena recordar que, como describimos en la revista **N.233**, el modo **indexado** funciona sumando la **dirección** de la **variable**, denominada también **desplazamiento** (**offset**), al valor contenido en el registro índice **X** o **Y**.

El ejemplo utilizado en aquella ocasión fue **loop clr (PROVS,x)**.

Como **PROVS** fue definida en la dirección **8Bh** y el registro **X** tenía el valor **9h**, la instrucción **clr** borra el byte contenido en la dirección **8Bh + 9h** y es decir en **94h**.

En el ejemplo que presentamos a continuación hemos realizado un programa donde en puntos diferentes se lanzan **subrutinas** de **conversión A/D**, cuyos resultados son almacenados en **dos variables** diferentes y los tres bytes siguientes a cada una.

Por tanto, en primer lugar **definimos** las **variables** en **Data Memory**, como en el ejemplo anterior, pero con una pequeña modificación.

Primero definimos un área de **4 bytes** asociando la dirección del primer byte a la variable **VALAD1**, a continuación definimos un área de **4 bytes** asociando la dirección del primer byte a la variable **VALAD2**. Simplificando podemos decir que **VALAD1** y **VALAD2** son **variables de 4 bytes**.

```
(0087h)     VALAD1      DS.B 4
(008Bh)     VALAD2      DS.B 4
.....       .........   .........
(0094h)     PUNTAT      DS.B 1
```

Ahora escribimos las **instrucciones** del programa de ejemplo:

```
(FA13h)             ld      a,#VALAD2
(FA15h)             ld      PUNTAT,a
(FA17h)             clr     x
(FA18h)             call    caricad
.....       .....   .....   .....
(FAD0h)             ld      a,#VALAD1
(FAD2h)             ld      PUNTAT,a
(FAD4h)             clr     x
(FAD5h)             call    caricad
.....       .....   .....   .....
(FAE2h)     caricad
.....       .....   .....   .....
(FAE9h)             btjf    ADCCSR,#7,$
(FAFCh)             ld      a,ADCDR
(FAFFh)             ld      ([PUNTAT],x),a
(FB02h)             inc     x
(FB03h)             cp      x,#4
(FB05h)             jrc     caricad
(FB07h)             ret
```

El primer grupo de cuatro instrucciones:

```
(FA13h)     ld      a,#VALAD2
(FA15h)     ld      PUNTAT,a
(FA17h)     clr     x
(FA18h)     call    caricad
```

es prácticamente igual al del ejemplo anterior. La única diferencia es la inserción de la instrucción:

```
(FA17h)     clr     x
```

con la que se **borra** (**clr**) el registro **X**.

La situación antes de la ejecución de la subrutina **caricad** es la siguiente:

- El registro **X** contiene **0**.
- La variable **PUNTAT** contiene el valor **8Bh**, que es la dirección de inicio de **VALAD2**.

La instrucción:

```
(FA18h)     call    caricad
```

lanza la subrutina:

```
(FAE2h)     caricad
.....       .....   .....   .....
(FAE9h)             btjf    ADCCSR,#7,$
(FAFCh)             ld      a,ADCDR
(FAFFh)             ld      ([PUNTAT],x),a
(FB02h)             inc     x
(FB03h)             cp      x,#4
(FB05h)             jrc     caricad
(FB07h)     ret
```

Las instrucciones de la subrutina son las mismas que las utilizadas en el ejemplo anterior, hasta llegar a la instrucción que utiliza el modo **indexado indirecto corto**:

```
(FAFFh)     ld      ([PUNTAT],x),a
```

Como en el ejemplo anterior, el acumulador **A** contiene el valor de la conversión A/D, es decir **1Fh**. Al trabajar en modo **indexado** la dirección de la variable constituye el desplazamiento al que hay que añadir el valor contenido en el registro **X**.

El registro índice **X** contiene el valor **0** y la variable **PUNTAT** está encerrada entre corchetes **[ ]**, por lo tanto no se utiliza su dirección de su definición (**94h**) sino el valor

contenido en esta dirección, es decir **8Bh**, que es la dirección de **VALAD2**.

El resultado de la ejecución de esta instrucción es que, la **primera vez** que se ejecuta la subrutina, el valor **1Fh** contenido en el acumulador **A** se almacena en la dirección de memoria **8Bh + 0** (es decir **VALAD2+0**).

Hemos indicado "primera vez" ya que, como se muestra en el grupo de instrucciones que exponemos a continuación, la **subrutina** se ejecuta **cuatro veces** antes de volver al programa principal.

```
(FB02h)     inc     x
(FB03h)     cp      x,#4
(FB05h)     jrc     caricad
(FB07h)     ret
```

En efecto, con la instrucción **inc x** se incrementa en **1** el valor del registro **X**, por lo tanto la **segunda vez** no contiene el valor **0** sino **1**.

Con la instrucción **cp x,#4**, el valor contenido en el registro **X** se compara con el valor inmediato **4**, caracterizado por el símbolo **#**. Si el valor es **menor** que **4** el **flag de Acarreo** (**Carry**) se pone a **1**, si es **mayor o igual a 4** se pone a **0**.

Con la instrucción siguiente (**jrc caricad**) el programa salta a la subrutina **caricad** si el **flag de Acarreo** vale **1** (**j**ump **r**elative **c**arry), si vale **0** se ejecuta la instrucción siguiente, que es la instrucción de **retorno** de subrutina (**ret**).

Ya que en este momento el valor contenido en el registro **X** es **1**, es decir menor que **4**, el **flag de Acarreo** vale **1** y, por lo tanto, el programa ejecuta de nuevo la parte de la subrutina relativa a la conversión A/D, hasta llegar a la instrucción:

```
(FAFFh)     ld      ([PUNTAT],x),a
```

Esta vez el nuevo valor convertido, disponible en el acumulador **A**, se almacena en la dirección **8Bh + 1** o (es decir **VALAD2+1**).

Posteriormente se vuelve a **incrementar** el registro **X**, pasando a valer **2**, y a compararse con **4**. Ya que este valor es **menor** que **4** el **flag de Acarreo** vale **1** y permite de nuevo la ejecución de la subrutina. En consecuencia, el valor de la conversión se almacena en **8Bh + 2** (**VALAD2+2**).

La siguiente vez se repetirá el proceso almacenándose el valor convertido en **8Bh + 3** (**VALAD2+3**). Ahora bien, cuando el registro **X**, a fuerza de ser incrementado, llega a contener el mismo valor con el que se compara, es decir **4**, lo que sucede después de **4** ciclos de ejecución de la **subrutina**, el **flag de Acarreo** se pone a **0**.

Dado que la condición de salto a la subrutina **caricad** (**jrc**) **no** se cumple, el programa continúa con la instrucción **ret**, volviendo así al programa principal, ejecutándose por tanto las restantes instrucciones:

```
(FAD0h)     ld      a,#VALAD1
(FAD2h)     ld      PUNTAT,a
(FAD4h)     clr     x
(FAD5h)     call    caricad
```

Resumiendo, el programa va almacenando el resultado de la conversión A/D en las siguientes direcciones:
87h + 0 (VALAD1+0)
87h + 1 (VALAD1+1)
87h + 2 (VALAD1+2)
87h + 3 (VALAD1+3)

Utilizando el modo **indexado indirecto corto** hemos podido direccionar áreas diferentes y consecutivas de memoria escribiendo una única subrutina de almacenamiento. En este modo los corchetes que caracterizan el direccionamiento **indirecto** se escriben en los extremos de una **variable** (denominada **desplazamiento**), que puede tener un valor máximo de **FFh** al tener un **byte** de longitud (**corta**).

INDIRECTO LARGO (LONG)

El modo **indirecto** se define como **largo** (**long**) cuando direcciona áreas de memoria superiores a **FFh** utilizando como operando una **variable de dos bytes** de longitud. En realidad hablar de variable de **dos bytes** no es del todo cierto sino una simplificación, ya que en el lenguaje **Assembler para ST7** no existe la posibilidad de asociar una **longitud** a las variables definidas. La **variable** siempre identifica únicamente la **dirección de memoria** en la que ha sido definida,

y como este lenguaje es de un microcontrolador con tecnología de **8 bits**, las instrucciones tratan valores contenidos en **1 byte**.

Entonces ¿qué quiere decir y cómo funciona el direccionamiento **largo** (**long**)?

Cuando se habla de una variable de **dos bytes** de longitud, como por ejemplo:

```
(0094h)     PUNTAT   DS.B 2
```

se **definen 2 bytes** (**DS.B 2**) y se **asocia** a la **variable** (**PUNTAT**) la **dirección de memoria** (**0094h**) del **primer byte** (nótese que **no** se asocian dos bytes a la variable ya que no es posible tal operación).

La directiva **DS.B** quiere decir "**D**efine **S**pace **B**ytes", y el número que la sigue indica el **número de bytes** de "espacio" definido o, mejor dicho, **reservados**. Si quisiéramos ahora definir otra variable, la siguiente dirección de memoria sería: **0094h + 2**, es decir **0096h**.

De esta forma reservamos un área de memoria de **2 bytes** a partir de la dirección **0094h**. A esta dirección hemos asociado la etiqueta **PUNTAT**.

Para explicar el direccionamiento **indirecto largo** vamos a utilizar un programa de ejemplo que incluye una subrutina que efectúa una serie de operaciones obteniendo los datos de una tabla de valores constantes. Esta subrutina se llama desde diferentes puntos del programa, utilizando valores contenidos en tablas diferentes.

En primer lugar definimos una **variable** que nos sirve para este tipo de direccionamiento:

```
(0094h)     PUNTAT   DS.B 2
```

A continuación definimos las **tablas** que contienen los valores: **3 tablas**, cada una con **4 elementos** y con valores predefinidos diferentes.

```
(FC03h)     TB_01    DC.B    01h,03h,05h,07h
(FC07h)     TB_02    DC.B    02h,04h,06h,08h
(FC0Ch)     TB_03    DC.B    10h,20h,30h,40h
```

Para definir las tablas hemos utilizado la directiva **DC.B**. En este caso los valores situados a la derecha de **DC.B** no definen el tamaño reservado en bytes sino el **valor** contenido en cada byte a partir de la **primera dirección** de **memoria** asociada a la etiqueta de la tabla.

En otras palabras, considerando la primera tabla, a la dirección de memoria **FC03h** le hemos asociado la etiqueta **TB_01** y hemos insertado en este byte el valor **01h**. Luego, en el byte siguiente, es decir en la dirección **FC04h**, hemos insertado el valor **03h**, y así sucesivamente.

A continuación escribimos las **instrucciones** del programa de ejemplo:

```
(FA13h)             ld      a,#TB_02.h
(FA15h)             ld      PUNTAT,a
(FA18h)             ld      a,#TB_02.l
(FA1Bh)             ld      {PUNTAT+1},a
(FA1Eh)             call    CALCOLT
.............  ..............  ..............  ..............
(FA23h)             ld      a,#TB_01.h
(FA25h)             ld      PUNTAT,a
(FA28h)             ld      a,#TB_01.l
(FA2Bh)             ld      {PUNTAT+1},a
(FA2Eh)             call    CALCOLT
.............  ..............  ..............  ..............
.............  ..............  ..............  ..............
(FA30h)             ld      a,#TB_03.h
(FA32h)             ld      PUNTAT,a
(FA35h)             ld      a,#TB_03.l
(FA38h)             ld      {PUNTAT+1},a
(FA3Bh)             call    CALCOLT
.............  ..............  ..............  ..............
.............  ..............  ..............  ..............
(FB51h)     CALCOLT ld      a,[PUNTAT.w]
.............  ..............  ..............  ..............
.............  ..............  ..............  ..............
(FBD4h)             ret
```

Con la primera instrucción, que utiliza direccionamiento **inmediato** (caracterizado por la presencia del símbolo **#**):

```
(FA13h)             ld      a,#TB_02.h
```

cargamos en el acumulador **A** el valor correspondiente a la dirección en la que **TB_02** está definida.

Esta tabla está definida en la dirección **FC07h**, que es de **dos bytes**. Ahora bien, el acumulador **A** solo puede contener un byte, es decir a lo sumo el valor **FFh**. ¿Qué hacer?.

Veamos en detalle el Assembler para ST7.

Seguramente todos habréis apreciado que la después de **TB_02** aparece "**. h**". En este caso **no** se hace referencia a notación hexadecimal, sino que indica que se toma en consideración el valor **high** de la dirección de **TB_02**, es decir el **byte más significativo**. Al procesarse la instrucción, en el registro acumulador **A** se carga, por tanto, el valor **FC**.

Con la instrucción siguiente:

```
(FA15h)        ld        PUNTAT,a
```

guardamos el valor **FC** en la variable **PUNTAT**, es decir en la dirección **0094h**.

El programa continúa con la instrucción:

```
(FA18h)        ld        a,#TB_02.l
```

En este caso **TB_02** termina con "**.l**", que indica que se toma en consideración el valor **low** de la dirección de **TB_02**, es decir el **byte menos significativo**. Al procesarse la instrucción, en el registro acumulador **A** se carga, por tanto, el valor **07**.

Resulta evidente que de este modo se "divide" la dirección de **TB_02** en **dos bytes** independientes.

Evidentemente antes de utilizar los **dos bytes** tenemos que guardar también este byte, pero siguiendo un **orden** preciso: El byte menos significativo debe guardarse en la dirección siguiente a la que hemos salvado el byte anterior. Esta operación se realiza con la instrucción siguiente:

```
(FA1Bh)        ld        {PUNTAT+1},a
```

Las dos **llaves { }** que encierran el operando hacen referencia al uso de una expresión. En nuestro caso queremos cargar el valor contenido en el acumulador **A** en la dirección **PUNTAT+1**.

El Compilador Assembler procesará esta expresión y generará un ejecutable donde la dirección del operando se convierte en **0095h**, que es precisamente la dirección del byte siguiente a **PUNTAT**.

Seguramente alguien se esté preguntado por qué hemos utilizado este método, ya que podríamos haber definido inicialmente:

```
(0094h)    PUNTAT1  DS.B 1
(0095h)    PUNTAT2  DS.B 1
```

y así, en vez de escribir:

```
(FA1Bh)        ld        {PUNTAT+1},a
```

podríamos haber escrito:

```
(FA1Bh)        ld        PUNTAT2,a
```

obteniendo el mismo resultado. Nosotros creemos más conveniente el método de la "**variable de 2 bytes**" ya que así los datos almacenados siempre son consecutivos. Utilizando "**dos variables de 1 byte**" se corren más riesgos al poder cometer algún error con más facilidad al acceder a los datos y la interpretación de lo que realiza el programa es algo más compleja.

Después de esta aclaración volvemos a nuestro ejemplo, donde encontramos la instrucción siguiente:

```
(FA1Eh)        call      CALCOLT
```

que llama a la subrutina:

```
(FB51h)    CALCOLT ld        a,[PUNTAT.w]
.....      ..... ..... .....
.....      ..... ..... .....
(FBD4h)        ret
```

En esta subrutina la primera instrucción utiliza direccionamiento **indirecto largo**:

```
(FB51h)    CALCOLT ld        a,[PUNTAT.w]
```

En efecto, la variable **PUNTAT** está encerrada entre corchetes **[ ]**, indicando un direccionamiento **indirecto**. "**.w** "(**word**, **2 bytes**) identifica el modo **largo**.

Con este direccionamiento se toman en consideración el byte contenido en la dirección de la variable **PUNTAT** y el **byte siguiente**. Dado que cargamos en estas direcciones los valores **FCh** y **07h**, en el registro **A** se almacena el valor presente en la dirección de memoria **FC07h**, es decir **02h**.

Para que quede claro lo anteriormente expuesto volvemos a recordar la declaración de la tabla en la dirección **FC07h**, así se puede constatar fácilmente que en **A** se carga el primer valor asociado a la dirección, es decir **02h**:

```
(FC07h)     TB_02    DC.B    02h,04h,06h,08h
```

El resto de instrucciones de la subrutina procesarán este dato, llegando a la última instrucción:

```
(FBD4h)              ret
```

que devuelve el control al programa principal, ejecutándose las instrucciones siguientes que cargan en los dos bytes reservados a la variable **long PUNTAT** el valor **FC03h**:

```
(FA23h)              ld      a,#TB_01.h
(FA25h)              ld      PUNTAT,a
(FA28h)              ld      a,#TB_01.l
(FA2Bh)              ld      {PUNTAT+1},a
(FA2Eh)              call    CALCOLT
```

Mediante la ejecución de la subrutina **CALCOLT** el valor almacenado en el acumulador **A** es **01h**:

```
(FC03h)     TB_01    DC.B    01h,03h,05h,07h
```

Las instrucciones siguientes:

```
(FA30h)              ld      a,#TB_03.h
(FA32h)              ld      PUNTAT,a
(FA35h)              ld      a,#TB_03.l
(FA38h)              ld      {PUNTAT+1},a
(FA3Bh)              call    CALCOLT
```

cargan en los dos bytes reservados a la variable **long PUNTAT** el valor **FC0Ch**. Ahora, mediante la ejecución de la subrutina **CALCOLT**, el valor almacenado en el acumulador **A** es **10h**:

```
(FC0Ch)     TB_03    DC.B    10h,20h,30h,40h
```

Resumiendo, se utiliza un direccionamiento **indirecto largo** (**long**) cuando se reserva al operando un área de memoria de **2 bytes** de longitud.

INDEXADO INDIRECTO LARGO (LONG)

En el ejemplo anterior hemos "apuntado" a tablas diferentes y con la misma subrutina hemos obtenido el **primer valor** de cada tabla. Si hubiéramos querido utilizar el **segundo**, el **tercero** o el **cuarto** valor ... ¿Qué tendríamos que hacer?

Muy sencillo: utilizar el modo **Indexado Indirecto Largo**. Se trata de un modo de direccionamiento análogo al anterior, pero con la inclusión de un **registro índice**, de ahí el término **indexado**. Introduciendo un valor en este registro podemos responder a la pregunta planteada anteriormente.

Por sencillez vamos a reutilizar el ejemplo anterior, que reproducimos a continuación modificando únicamente la subrutina **CALCOLT**.

En primer lugar definimos una **variable**:

```
(0094h)     PUNTAT   DS.B 2
```

A continuación definimos las **tablas** que contienen los valores: **3 tablas**, cada una con **4 elementos** y con valores predefinidos diferentes.

```
(FC03h)     TB_01    DC.B    01h,03h,05h,07h
(FC07h)     TB_02    DC.B    02h,04h,06h,08h
(FC0Ch)     TB_03    DC.B    10h,20h,30h,40h
```

Ahora escribimos las **instrucciones** del programa de ejemplo:

```
(FA13h)              ld      a,#TB_02.h
(FA15h)              ld      PUNTAT,a
(FA18h)              ld      a,#TB_02.l
(FA1Bh)              ld      {PUNTAT+1},a
(FA1Eh)              call    CALCOLT
.....   ..........   .....   .....
.....   ..........   .....   .....
(FA23h)              ld      a,#TB_01.h
(FA25h)              ld      PUNTAT,a
(FA28h)              ld      a,#TB_01.l
(FA2Bh)              ld      {PUNTAT+1},a
(FA2Eh)              call    CALCOLT
.....   ..........   .....   .....
.....   ..........   .....   .....
(FA30h)              ld      a,#TB_03.h
(FA32h)              ld      PUNTAT,a
(FA35h)              ld      a,#TB_03.l
(FA38h)              ld      PUNTAT+1},a
(FA3Bh)              call    CALCOLT
.....   ..........   .....   .....
.....   ..........   .....   .....
(FB51h)     CALCOLT  ld      x,#0FFh
(FB53h)     LOOPX    inc     x
(FB54h)              ld      a,([PUNTAT.w],x)
.....   ..........   .....   .....
(FBD0h)              cp      x,#3
(FBD2h)              jrne    LOOPX
(FBD4h)     ret
```

Como ya hemos explicado, después de ejecutar las instrucciones:

```
(FA13h)   ld     a,#TB_02.h
(FA15h)   ld     PUNTAT,a
(FA18h)   ld     a,#TB_02.l
(FA1Bh)   ld     {PUNTAT+1},a
(FA1Eh)   call   CALCOLT
```

La variable **PUNTAT** "contiene" la dirección de **TB_02**, es decir **FC07h**.

Cuando se lanza la subrutina:

```
(FB51h)   CALCOLT ld     x,#0FFh
```

en el registro índice **X** se carga el valor **255** (**FFh**) que, como ya sabéis, es el valor máximo que puede contener un registro de **8 bits**.

Con la instrucción:

```
(FB53h)   LOOPX   inc    x
```

se incrementa en **1** el valor contenido en el registro **X**. Además esta instrucción está identificada con la etiqueta **LOOPX**. Puesto que el registro **X** contiene el máximo valor permitido, es decir **255**, este incremento hace que el valor del registro **X** pase a **0**, además de ponerse a **1** el **flag de Acarreo**.
La instrucción siguiente:

```
(FB54h)   ld     a,([PUNTAT.w],x)
```

es una instrucción que utiliza el **modo indirecto** (caracterizado por el uso de **corchetes**), **indexada** (utiliza el registro índice **X** encerrado entre **paréntesis**) y de tipo **largo** (**. w**). Con este modo de direccionamiento la dirección de la variable constituye el **desplazamiento** (**offset**) al que se le añade el valor contenido en el registro **X**.

La **primera vez** que se ejecuta la **subrutina** en el acumulador **A** se carga el valor contenido a la dirección de **[PUNTAT.w]**, es decir **FC07h**, que, como hemos explicado, es **02h**, más el valor del registro **X**, que es **0**. En resumen, la primera vez se almacena en **A** el valor **02h**.

Hemos indicado "primera vez" ya que las instrucciones siguientes:

```
(FBD0h)   cp     x,#3
(FBD2h)   jrne   LOOPX
```

ejecutan la subrutina hasta que el valor contenido en el registro **X** es diferente de **3**. En efecto primero se **compara** (**cp**) con **3** y si es diferente el programa **salta** (**jrne**) a la etiqueta **LOOPX**, donde se **incrementa** (**inc**) en **1** el registro **X**.

NOTA: La instrucción **jrne** es el acrónimo de **J**ump **R**elative if **N**ot **E**qual, es decir salta si no es igual.

Cada vez que se produce un salto a la etiqueta **LOOPX** se añade a la dirección del desplazamiento, que es siempre **FC07h**, un valor de **X incrementado**. Por tanto, en el acumulador **A** se van cargando **todos** los **valores de la tabla**, como detallamos a continuación:
Recordemos que la definición de la tabla **TB_02** fue:

```
(FC07h)   TB_02   DC.B   02h,04h,06h,08h
```

Por tanto, como ya hemos expuesto, la primera vez que se ejecuta la instrucción:

```
(FB53h)   LOOPX   inc    x
```

X = 0
FC07h + 0
se carga en **A** el valor **02h**.
La segunda vez:

```
(FB53h)   LOOPX   inc    x
```

X = 1
FC07h + 1
se carga en **A** el valor **04h**.
La tercera vez:

```
(FB53h)   LOOPX   inc    x
```

X = 2
FC07h + 2
se carga en **A** el valor **06h**.
La cuarta y última vez:

```
(FB53h)   LOOPX   inc    x
```

X = 3
FC07h + 3
se carga en **A** el valor **08h**.

Cuando el registro **X** contiene el valor **3** la condición de salto **no** se cumple, por lo tanto el programa continúa con la instrucción siguiente:

```
(FBD4h)   ret
```

lo que provoca la salida de la subrutina y la vuelta al programa principal, donde se ejecutan instrucciones similares para las tablas **TB_01** y **TB_03**.

RESUMEN

Direccionamiento Indirecto

En las instrucciones que utilizan direccionamiento **indirecto** la variable de acceso a memoria **no** contiene el **operando** sino la **dirección** que contiene el **operando**, escribiéndose siempre entre **corchetes [ ]**.

En estas instrucciones el operando puede ser tanto la **fuente** como el **destino** del resultado de la instrucción.

En modo **corto** (**short**) se puede acceder a una dirección de memoria incluida entre **00h** y **FFh**. En modo **largo** (**long**), caracterizado por la presencia de "**. w**", se puede acceder a una dirección de memoria incluida entre **0000h** y **FFFFh** (ver Tabla N.1).

Direccionamiento Indexado Indirecto

Este modo de direccionamiento es similar al **indirecto** con la diferencia de que además utiliza los **registros índice X** e **Y** para acceder a una dirección de memoria.

En las instrucciones con este modo de direccionamiento, los **paréntesis ( )** situados en los extremos del operando, que puede ser tanto fuente como destino, indican la utilización del modo **indexado**, mientras que los **corchetes [ ]** situados en los extremos de la variable indican la utilización del modo **indirecto**.

Para direccionar en este modo se utiliza una **variable** (**desplazamiento**) y un **registro índice** (**X** o **Y**), separados por una **coma**.

Se trabaja en modo **corto** (**short**) cuando la dirección del **desplazamiento** es de **1 byte**. Al sumar el valor contenido en el registro **X** o **Y** la instrucción puede direccionar un área de memoria incluida entre **00h** y **1FEh**.

Se trabaja en modo **largo** (**long**) cuando la dirección del **desplazamiento** es de **2 bytes** (**1 word**), estando presente la indicación "**.w**". Al sumar el valor contenido en el registro **X** o **Y** la instrucción puede direccionar un área de memoria incluida entre **0000h** y **FFFFh**.

Ejemplos de Códigos de Operación

Como colofón a este artículo presentamos algunos ejemplos de instrucciones en **formato Assembler** y en **formato ejecutable**, es decir el **código de operación** (**op-code**). Las abreviaturas que hemos utilizado son las mismas que las presentes en los manuales de los fabricantes de micros ST7. Su significado se presenta a continuación.

[short] . . . . indirecto corto
[long]. . . . . indirecto largo
([short],x) . . indexado indirecto corto
([long],x). . . indexado indirecto largo

MODO	INSTRUCCIÓN	OP-CODE
[short]	ld a,[punt]	92 B6 80
[long]	ld a,[punt.w]	92 C6 80
([short],x)	ld a,([punt],x)	92 E6 80
([long],x)	ld a,([punt.w],x)	92 D6 80
([short],y)	ld a,([punt],y)	91 E6 80
([long],y)	ld a,([punt.w],y)	91 D6 80

92 indica **indirecto** / **indexado indirecto** con **X**
B6 indica modo **corto** (short)
C6 indica modo **largo** (long)
E6 indica modo **indexado corto** (short)
D6 indica modo **indexado largo** (long)
91 indica **indexado indirecto** con **Y**
80 Dirección de ejemplo de la variable **punt**

TABLA N.1

Modo	Ejemplo de formato		Memoria direccionada	Desplazamiento
Indirect Short	ld a,[0E4h]	ld a,[pippo]	00h-FFh	1 Byte
Indirect Long	ld [3Ch.w],a	ld [pippo.w],a	0000h-FFFFh	1 Word
Indirect Indexed Short	ld ([96h],x),a	ld ([pippo],x),a	00h-1FEh	1 Byte
Indirect Indexed Long	ld ([3Ch.w],x),a	ld ([pippo.w],x),a	0000h-FFFFh	1 Word

En esta tabla hemos incluido dos ejemplos de cada uno de los modos de direccionamiento tratados en este artículo, incluyendo el área de memoria que se puede direccionar con cada modo.

TABLA N.2 INSTRUCCIONES Y DIRECCIONAMIENTO (TERCERA PARTE)

Mnemo Instrucción	Descripción Instrucción	Direccionamiento			
		[short]	[long]	([short],X)	([long],X)
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	•	•	•	•

En la Tabla N.2 se muestran las instrucciones que pueden direccionarse con los modos de direccionamiento indirecto e indexado indirecto, tanto en modo corto (short) como en modo largo (long).