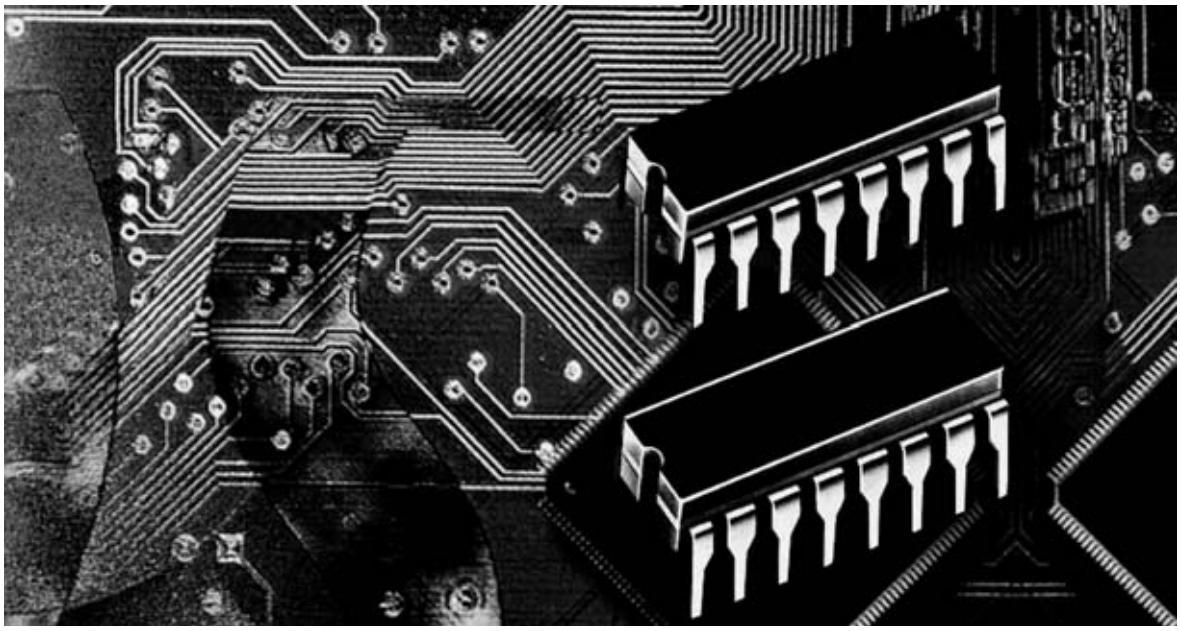

Programación con microcon

Con este artículo concluimos la exposición, iniciada en la revista N.233, de los distintos modos de direccionamiento de los micros ST7. Se trata de cuatro sencillos modos agrupados bajo el nombre de BIT OPERATION que tienen en común que su operando es un bit en lugar de uno o dos bytes.

Ciertamente los modos de direccionamiento del lenguaje Assembler en general, y los modos de los microcontroladores ST7 en particular, son una herramienta muy potente para acceder de diversas formas a las estructuras de datos. No obstante la gran cantidad de modos hace que su exposición detallada sea relativamente larga.

Si nos habéis seguido desde las primeras entregas, sabéis que en los modos tratados hasta ahora el dato implicado siempre era un **valor entero** contenido en variables o registros. En este artículo vamos a desarrollar **cuatro** modos cuya **característica** principal consiste en que las operaciones se efectúan sobre un **bit individual**:

BIT DIRECTO
BIT INDIRECTO
BIT RELATIVO DIRECTO
BIT RELATIVO INDIRECTO

Analizando los ejemplos que hemos preparado constatareis que estos modos son muy parecidos a los modos de los direccionamiento que ya hemos tratado, es decir los modos **directos, indirectos** y **relativos.**

De hecho en algunos manuales no se hace distinción entre los modos **directo** y **bit directo** o entre **indirecto** y **bit indirecto.** Nosotros preferimos marcar esta distinción para mostrar las cuestiones específicas de cada modo y analizarlos de forma exhaustiva.

Los **modos** que vamos a tratar son únicamente de tipo **short** (corto), lo que implica que los operandos han de estar definidos entre **00h** y **FFh.** Dadas las características de estos cuatro modos se utilizan exclusivamente con **instrucciones** que desarrollan **operaciones** con **bits** o que prueban un **bit** determinado para realizar un **salto relativo.**

Las cuatro **instrucciones** que pueden utilizar los cuatro modos BIT OPERATION son las siguientes: **BRES (Bit RESet):** Pone a 0 el **bit** indicado en el operando.

BSET (Bit SET): Pone a **1 el bit** indicado en el operando.

BTJF (Bit Test and Jump if False): **Analiza** el **bit** indicado en el operando y **salta** si es igual a **0.**

BTJT (Bit Test and Jump if True): Analiza el **bit** indicado en el operando y **salta** si es igual a **1.**

Antes de pasar a los ejemplos que hemos preparado para cada modo creemos oportuno recordar algunos conceptos para entender perfectamente el resto del artículo.

Carga en el acumulador **A** el **valor presente** en la dirección de memoria **38h.** En pocas palabras, podemos decir que cuando el símbolo **# (almohadilla)** precede a un **número** el operando es el propio número y no una dirección de memoria.

En realidad hay gran diferencia entre estos dos modos de direccionamiento, de hecho los códigos de operación en formato ejecutable **(op-code)** generados por el Compilador, el Linkador y el Montador son diferentes. El op-code de la instrucción **ld a,#38h** es **A638h,** dónde **A6** significa "carga en el acumulador A el valor presente en el operando" y **38h** es el valor inmediato a cargar. Por tanto, el valor a cargar es parte del código de operación ya que es el operando de la instrucción, y, como

troladores ST7 LITE 09 (6)

REPASO de los modos INMEDIATO y DIRECTO

Cuando, en la revista **N.233,** abordamos el direccionamiento **INMEDIATO** precisamos que este modo de direccionamiento utiliza como **operando** un **valor numérico** incluido entre **0** y **255 (00h** y **FFh).** La instrucción:

ld a,#38h

Carga el **valor 38h** en el acumulador **A.**
Analizando el direccionamiento **DIRECTO** vimos que el valor implicado es el contenido de la dirección de memoria del operando. Por tanto la instrucción:

ld a,38h

consecuencia, enseguida está disponible en la fase de ejecución del programa. En efecto, cuando el microcontrolador procede a ejecutar esta instrucción carga el valor **38h** en el acumulador **A.** En cambio el op-code de la instrucción **ld a,38h** es **B638h,** dónde **B6** significa "carga en el acumulador A el valor presente en la dirección de memoria indicada por el operando", es decir en la dirección **38h.** En este caso el valor no está inmediatamente disponible en el código de operación (op-code) sino que tiene que ser **decodificado** en la fase de **ejecución.**

Cuando el microcontrolador procede a ejecutar esta instrucción tiene que posicionarse previamen-

Artículos pulicados sobre los MODOS de DIRECCIONAMIENTO

Esta tabla sinóptica muestra la relación de artículos anteriormente publicados sobre **modos de direccionamiento** de las instrucciones **Assembler** de los microprocesadores de la **familia ST7** y los números de revista correspondientes.

Direccionamiento **inherente**, **inmediato**, **directo** e **indexado directo**	**Revista N.233**
Direccionamientos **relativos**	**Revista N.240**
Direccionamiento **indirecto** e **indexado indirecto**	**Revista N.241**

te en la dirección indicada por el operando y después mover al acumulador **A** el valor presente en la dirección de memoria. Una vez efectuado este pequeño repaso vamos a analizar los modos de direccionamiento BIT OPERATION.

BIT DIRECTO
Direccionamiento Directo de un Bit

Este modo utiliza **uno** de los ocho **bits** del valor contenido en la dirección de memoria implicada.

Como ejemplo de este modo de direccionamiento presentamos un programa que en determinados momentos realiza una serie de "intermitencias" en el terminal **2** del Puerto **A (PA2)** y en otros momentos en el terminal **2** del Puerto **B (PB2).** En el listado no incluimos la definición de los registros y de los puertos del micro. Recordamos que todas las definiciones están contenidas en el archivo ST72FL09.INC. Aquí nos basta con tener presente que el valor de definición del **Puerto A** es 00h y el valor del **Puerto B** es 03h. También recordamos que los valores **hexadecimales** en cursiva encerrados entre paréntesis corresponden a la teórica dirección de memoria de las instrucciones.

```
(FA0Ch)   principal   clr     PORT_A
                      clr     PORT_B
          inicio
.........   .........   .........   .........
                      call    lampa
.........   .........   .........   .........
                      call    lampb
.........   .........   .........   .........
                      jp      inicio

(FA5Bh)   lampa       bset    PORT_A,#2
                      bres    PORT_A,#2
                      ret
.........   .........   .........   .........
(FA68h)   lampa       bset    PORT_B,#2
                      bres    PORT_B,#2
                      ret
```

Las líneas con la instrucción clr borran el contenido de **PORT_A** y **PORT_B**.
Cuando la instrucción **call lampa** llama la subrutina **lampa:**

```
(FA5Bh)   lampa       bset    PORT_A,#2
```

La instrucción **bset** pone a **1** el bit **2** del **Puerto A** y, por consiguiente, el **diodo LED** conectado a este terminal se **enciende** si su Ánodo está conectado al terminal **2** y su Cátodo está conectado a masa, o bien se apaga si su Ánodo está conectado a **Vcc** y su Cátodo está conectado al terminal **2.**

El símbolo # indica que el 2 no es una dirección sino un **valor numérico,** que corresponde precisamente al **bit** que es puesto a 1. Aparentemente puede parecer una instrucción de tipo **inmediato,** ahora bien la instrucción es de tipo **DIRECTO** ya que, como aclararemos a continuación, el microcontrolador no puede ejecutarla sin previa decodificación. La explicación está en el **op-code** resultante de la compilación. El op-code de la instrucción es **1400h,** dónde **14** significa "poner a 1" el bit 2 y **00h** es la dirección de **PORT_A.**

NOTA: Para conocer el op-code del **bit** a **poner** a 1 el compilador Assembler utiliza la siguiente expresión **(en hexadecimal): 10 + (2 x nº bit).** Este es el motivo por el cual para poner a 1 el bit 2 las primeras cifras del op-code son **14h.** Si, por ejemplo, se hubiera querido poner a 1 el bit **3 (bset PORT_A,#3)** las primeras cifras del op-code habrían sido **16h.**

Como se puede deducir el valor disponible en el op-code no se puede utilizar inmediatamente, sino que tienen que ser **decodificado** por el microcontrolador para acceder a la dirección indicada por el operando, que es **00h** (dirección de **PORT_A).** Dado que el **valor** implicado, aunque sea solo **un bit,** está **contenido** en la dirección de memoria del operando, estamos en presencia de un modo de direccionamiento **directo,** aunque en la instrucción aparezca un signo #. Además, se trata de un modo **short** (corto), ya que permite utilizar en el operando variables definidas entre **00h** y **FFh.**

Una vez aclarado este punto, continuamos con la descripción del ejemplo. La siguiente instrucción **(bres PORT_A,#2)** pone a 0 el **bit 2** del **Puerto A** y, por consiguiente, el **diodo LED** conectado a este terminal se apaga si su Ánodo está conectado al terminal **2** y su Cátodo está conectado a masa, o bien se enciende si su Ánodo está conectado a **Vcc** y su Cátodo está conectado al terminal **2.** Con dos instrucciones **(bset y bres)** hemos escrito una subrutina que hace **parpadear** un **diodo LED** conectado a un terminal del **Puerto A.** La instrucción **ret** determina el retorno de la

subrutina, una vez ejecutada se vuelve al programa principal. Las instrucciones correspondientes a la segunda subrutina (etiqueta **lampb)** son exactamente iguales con la diferencia de que afectan a **PORT_B.**

Considerando que las dos subrutinas **lampa** y **lampb** tienen la misma función, aunque para dos puertos diferentes, hay que plantearse si no existe la posibilidad de unificarlas en **una** sola **subrutina** y **acceder** en un momento dado al **puerto** deseado. Con el modo de direccionamiento siguiente vamos a ver que esto es posible.

BIT INDIRECTO

Direccionamiento Indirecto de un Bit

Para exponer este modo vamos a utilizar el listado del ejemplo anterior, obviamente con alguna modificación.

```
(0086h)   PUNTAT     DS.B 1

(FA0Ch)   principal  clr      PORT_A
                     clr      PORT_B
          inicio
                     ld       a,#PORT_A
                     ld       PUNTAT,a
                     call     lampx
.........  .........  .........  .........
                     ld       a,#PORT_B
                     ld       PUNTAT,a
                     call     lampx
.........  .........  .........  .........
                     jp       inicio

(FA5Bh)   lampx      bset     [PUNTAT],#2
                     bres     [PUNTAT],#2
                     ret
```

Con la primera instrucción definimos la variable **PUNTAT** en la dirección de memoria **(Data RAM) 86h.** La instrucción **ld a,#PORT_A,** de direccionamiento **inmediato,** carga en el acumulador **A** la **dirección** de definición del registro **PORT_A,** que, como hemos explicado en el ejemplo anterior, es **00h.** La instrucción siguiente **(ld PUNTAT,a)** carga en la variable **PUNTAT** el valor contenido en el acumulador **A,** es decir **00h.**

Por último, la instrucción **call lampx** lanza la ejecución de la subrutina con etiqueta **lampx:**

```
(FA5Bh)   lampx      bset     [PUNTAT],#2
```

Como se puede observar la variable **PUNTAT** está encerrada entre corchetes, lo que indica que se trata de una instrucción de direccionamiento **Indirecto.** En este modo **no** se pone a 1 **(bset)** el **bit 2** del valor contenido en la variable **PUNTAT** sino el **bit 2** del valor contenido en la **dirección** de **memoria** a la que apunta la variable **PUNTAT,** es decir **00h,** que corresponde a la dirección de **PORT_A.**

De forma análoga opera la instrucción siguiente, **bres [PUNTAT],#2,** en este caso poniendo a 0 el bit implicado. La subrutina termina con la instrucción **ret.** El programa principal continúa con el siguiente grupo de instrucciones:

```
          ld       a,#PORT_B
          ld       PUNTAT,a
          call     lampx
```

Puesto que el registro **PORT_B** está definido en la dirección **03h** de **Data RAM,** antes de lanzar la subrutina **lampx** se carga en el variable **PUNTAT** la dirección de **PORT_B,** es decir **03h.** Cuando se ejecute la **subrutina lampx,** a través de la instrucción call lampx, las instrucciones de la subrutina afectarán al diodo LED conectado a la terminal PB2 ya que la variable **PUNTAT** contiene la dirección de memoria de **PORT_B.**

En efecto, como hemos explicado anteriormente, la instrucción:

```
(FA5Bh)   lampx      bset     [PUNTAT],#2
```

No **pone** a 1 el bit 2 del valor contenido en **PUNTAT** sino el **bit 2** del valor contenido a la **dirección** de **memoria** a la que apunta la variable **PUNTAT,** es decir **03h.** Por tanto se pone a 1 el bit **2** de **PORT_B.**

Por el mismo motivo con la instrucción siguiente:

```
          bres     [PUNTAT],#2
```

Se **pone** a 0 el bit **2** de **PORT_B.**
Por último, con la instrucción ret regresamos al programa principal.

Resumiendo, hemos utilizado la misma **subrutina (lampx)** para modificar el estado del **bit 2** del puerto que nos interesa en cada momento.

Seguramente llegado este momento alguien se plantee que se pueden reducir las instrucciones a una sola:

```
                bset    [PUNTAT],NUMBIT
```

Donde **NUMBIT** es una variable que contiene en su momento el bit a tratar. En este caso el compilador generará un **error** ya que el **número** del **bit** a **poner** a 1 solo puede expresamente en la forma numérica **#n.** El motivo es simple: Una misma instrucción no puede mover de memoria a memoria.

Volviendo nuevamente al primer listado de ejemplo, es decir al programa que utiliza las subrutinas **lampa** y **lampb,** vamos a analizar más en detalle las instrucciones de las dos subrutinas que hacen parpadear **diodos LED** conectados a los terminales **2** de los Puertos **A** y **B.** Para operar de forma adecuada es necesario que el tiempo de **"encendido"** sea razonablemente igual al tiempo de "apagado". Tal como hemos propuesto, las dos subrutinas no responden a este criterio ya que el tiempo de "encendido" es más breve que el tiempo de **"apagado".**

Podríamos insertar una serie de instrucciones, calculando con precisión los ciclos de duración, para realizar un retardo de tal forma que los tiempos fueran iguales. Seguramente penséis que se trata de una solución "un poco complicada" y si se puede hacer algo más sencillo para solucionar este problema. La respuesta es que sí se puede, utilizando el modo de direccionamiento que presentamos a continuación.

BIT RELATIVO DIRECTO

Direccionamiento Relativo Directo de un Bit

Para analizar este modo vamos a realizar unas pequeñas modificaciones:

```
(FA0Ch)  principal  clr     PORT_A
                    clr     PORT_B
         inicio
.....    .........  ......  ......
                    call    lampa
.....    .........  ......  ......
                    call    lampb
.....    .........  ......  ......
                    jp      inicio

(FA5Bh)  lampa      btjt    PORT_A,#2,la_as
                    bset    PORT_A,#2
                    ret
         la_as      bres    PORT_A,#2
                    ret
(FA68h)  lampb      btjt    PORT_B,#2,la_bs
                    bset    PORT_B,#2
                    ret
         la_bs      bres    PORT_B,#2
                    ret
```

Cuando el programa llega a call lampa ejecuta la subrutina lampa, procesando la instrucción:

```
(FA5Bh)  lampa      btjt    PORT_A,#2,la_as
```

El significado de esta instrucción es "salta a la etiqueta **la_as** si el **bit 2** del **Puerto A** es a **1** (true) en caso contrario continuar con la instrucción siguiente".

Por tanto, en el caso de que el **bit 2** del **Puerto A** sea **1** (true) el programa salta a la etiqueta **la_as** donde, mediante la instrucción **bres PORT_A,#2** se pone a **0** el **bit 2.** La instrucción ret devuelve el control al programa principal. En caso contrario, es decir si el **bit 2** está a **0,** el programa continúa con la instrucción **bset PORT_A,#2,** poniendo a **1** el **bit 2.** La instrucción **ret** devuelve el control al programa principal. De esta forma el **tiempo** que transcurre entre una ejecución de la subrutina lampa y la siguiente es constante. Las instrucciones correspondientes a la segunda subrutina (etiqueta **lampb)** son exactamente iguales con la diferencia de que afectan a **PORT_B.**

BIT RELATIVO INDIRECTO
Direccionamiento Relativo Indirecto de un Bit

Ya solo queda analizar el último modo. En este caso unificamos en una sola **subrutina (lampx)** las subrutinas expuestas en el modo anterior. Para ello vamos a modificar ligeramente el ejemplo utilizado en el direccionamiento **Bit Indirecto.**

```
(0086h)  PUNTAT     DS.B 1

(FA0Ch)  principal  clr     PORT_A
                    clr     PORT_B
                    bset    PORT_B ,#2
         inicio     ld      a,#PORT_A
                    ld      PUNTAT,a
                    call    lampx
.....    .........  ......  ......
                    ld       a,#PORT_B
                    ld      PUNTAT,a
                    call    lampx
.....    .........  ......  ......
                    jp      inicio
```

```
(FA5Bh)   lampx   btjt   [PUNTAT],#2,la_x
          bset   [PUNTAT],#2
          ret
          la_x   bres   [PUNTAT],#2
          ret
```

La instrucción **btjt** de la subrutina **lampx** permite obtener tiempos iguales de encendido y apagado analizando el **bit 2** de la dirección apuntada por la variable **PUNTAT,** que contiene la dirección de **PORT_A** o la de **PORT_B.**

Hemos añadido una instrucción para "animar" un poco el artículo. En efecto, al principio del programa, además de borrar con la instrucción clr ambos puertos, hemos añadido una instrucción posterior:

```
(FA0Ch)   principal   clr   PORT_A
          clr   PORT_B
          bset   PORT_B ,#2
```

Esta instrucción pone a 1 **(bset)** el **bit 2** de **PORT_B.** De esta forma desfasamos los tiempos de encendido y apagado de los dos puertos, es decir cuando un LED se enciende el otro se apaga, y viceversa.

RESUMEN

Los cuatro modos de direccionamiento presentados en este artículo se utilizan para **modificar** el **estado lógico** de un **bit** o para **analizar** un **bit,** provocando, dependiendo del estado de la condición, un **salto relativo.**
Dadas sus características este direccionamiento se utiliza exclusivamente con las instrucciones **bres, bset, btjf, btjt** (ver Tabla N.1) y en modo **short** (corto), es decir los operandos tienen que estar definidos entre **00h** y **FFh.** El número de **bit** contenido en el operando puede ser referenciado únicamente con la forma **#n,** dónde **n** es el **número** del bit que modificar o a consultar (**0** a **7**). Cuando el operando está encerrado entre corchetes **[]** el modo de direccionamiento es **indirecto.**

PRÓXIMOS ARTÍCULOS

En próximos artículos empezaremos a proponer **ejemplos prácticos** de programación utilizando las **tarjetas experimentales LX.1548** y **LX.1549** presentadas en la revista **N.228.** Entre otros argumentos también afrontaremos las **rutinas** para **administrar** las funciones del micro, como el **Temporizador (Timer)**, la **SPI** y el **Conversor A-D.**

TABLA N.1 INSTRUCCIONES Y DIRECCIONAMIENTO

Mnemo Instrucción	Descripción Instrucción	Direccionamiento short	Direccionamiento [short]
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		