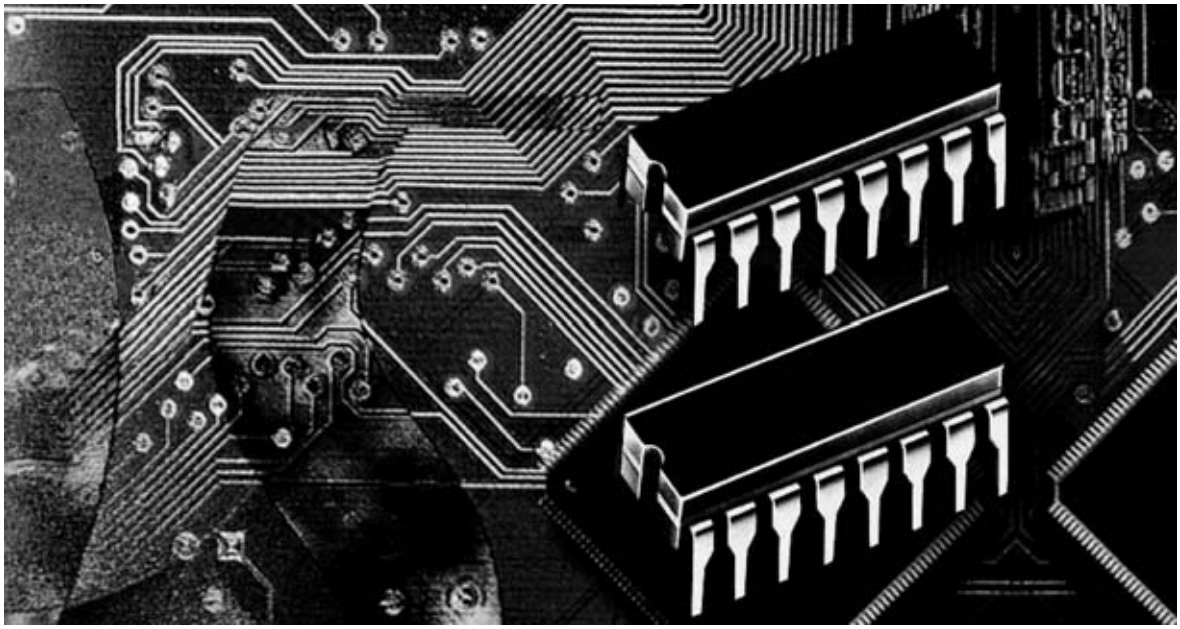

Programación con microcon

Con el artículo anterior de esta serie hemos concluido el amplio capítulo dedicado a los modos de direccionamiento. Ahora ha llegado el momento de afrontar otro tema muy extenso: El conjunto de las instrucciones Assembler soportadas por los micros ST7, que con sus 63 instrucciones permite realizar cualquier operación.

Llegado este punto de nuestro "curso" sobre el lenguaje Assembler para ST7 estáis familiarizados con la **lógica** y con la **técnica** de **direccionamiento** de los operandos. Como hemos visto, cada **instrucción** tiene muchas posibilidades de direccionamiento, por lo que con una misma instrucción se pueden conseguir resultados muy diferentes para ajustarse a cualquier necesidad.

Ahora ha llegado el momento de afrontar el juego de **instrucciones Assembler** para el microprocesador **ST7**, analizando las funciones que realizan cada una de las instrucciones.

El tema que ahora comenzamos es, sin duda, bastante largo, si bien una vez que se conocen todos los modos de direccionamiento es inevitable afrontarlo. El conjunto de instrucciones **Assembler** consta de **63 instrucciones**, siendo, por múltiples motivos, imposible exponerlo en un único artículo si se quiere hacer, como es nuestra línea, de forma sencilla, exhaustiva y con múltiples ejemplos clarificadores.

Para exponer las instrucciones hemos preferido, en lugar del típico orden **alfabético**, una agrupación en base a sus **funciones**. De esta forma resultará más sencillo y eficaz para nuestras explicaciones y para vuestra comprensión. No obstante, para que podáis hacer **consultas rápidas** hemos reproducido en primer lugar una **tabla** con el **listado alfabético** de las **instrucciones Assembler**. En la primera columna (**mnemónico**) se representa el **código** de la propia instrucción, en la segunda columna su **significado** (en inglés ya que el código de la instrucción procede de su significado en inglés) y en la tercera columna su **descripción** en español.

El método que hemos adoptado para analizar las **63 instrucciones** que componen el conjunto de instrucciones Assembler para los micros de la familia **ST7** es subdividirlas en base al tipo de función que realizan.

Cuando afrontemos los diferentes grupos, para cada instrucción proporcionaremos una definición detallada de sus **funciones**, la **sintaxis** (relaciones entre la instrucción y los operandos), el **formato** (forma adecuada de escribir la instrucción), los tipos de **direccionamiento** que soporta, el **op-code** (codificación en formato ejecutable), los **ciclos maquina** (tiempo necesario para su ejecución), la **longitud** en bytes de la instrucción y los **Flags afectados**. También exponemos una **leyenda** con las **abreviaturas utilizadas**. En este caso hemos respetado las mismas que las utilizadas en los manuales Assembler de los fabricantes.

Leyenda de Abreviaturas

dst = operando destino de la instrucción
src = operando fuente de la instrucción
reg = registro A o X o Y
mem = dirección de memoria

A = registro Acumulador
X = registro índice X
Y = registro índice Y
PC = registro Program Counter
S = registro Stack Pointer
CC = registro Condition Code
H = bit 4 Half Carry flag del CC
I = bit 3 Interrupt Mask flag del CC
N = bit 2 Negative flag del CC
Z = bit 1 Zero flag del CC
C = bit 0 Carry flag del CC

XX = valor del operando
MS = most significant, valor más significativo
LS = least significant, valor menos significativo

troladores ST7 LITE 09 (7)

SUBDIVISIÓN en GRUPOS de las intrucciones Assembler ST7

1° grupo	Instrucciones de carga	clr	ld					
2° grupo	Instrucciones de pila	pop	push	rsp				
3° grupo	Instrucciones de incremento-decremento	dec	inc					
4° grupo	Instrucciones de comparación - test	bcp	cp	tnz				
5° grupo	Instrucciones con operaciones lógicas	and	cpl	neg	or	xor		
6° grupo	Instrucciones con operaciones de bit	bres	bset	btjf	btjt			
7° grupo	Instrucciones aritméticas	adc	add	mul	sbc	sub		
8° grupo	Instrucciones rotación-desplazamiento	rlc	rrc	sla	sll	sra	srl	swap
9° grupo	Instrucciones de salto no condicional	call	callr	jp	jra	nop	ret	
10° grupo	Instrucciones de salto condicional	jr??						
11° grupo	Gestión de Interrupciones	halt	iret	trap	wfi			
12° grupo	Gestión del registro Condition Code	rcf	rim	scf	sim			

NOTA: Por motivos de espacio no hemos escrito todas las instrucciones de **salto condicional**, caracterizadas por la forma **JR??**, dónde **??** define la condición para el salto. En la lista de las instrucciones en orden alfabético sí se encuentran estas instrucciones y, por supuesto, quedarán definidas ampliamente cuando tratemos las instrucciones de salto condicional.

TODAS las instrucciones ASSEMBLER para ST7

Neumó.	Descripción	Definición
adc	Addition with Carry	Suma en A el valor de A más el fuente más el Flag C
add	Addition	Suma en A el valor de A más el fuente
and	Logical And	Realiza una operación AND entre A y el fuente, lo deja en A
bcp	Logical Bit compare	Compara A con el fuente
bres	Bit reset	Pone a cero el bit especificado
bset	Bit set	Pone a 1 el bit especificado
btjf	Bit test and Jump if false	Salta si el bit especificado es cero
btjt	Bit test and Jump if true	Salta si el bit especificado es 1
call	Call subroutine	Salta a la dirección destino y salva el PC en la pila
callr	Call subroutine relative	Salta a la dirección destino y salva el PC en la pila
clr	Clear	Pone a 00 el byte destino
cp	Compare	Compara el valor del fuente con A o X o Y
cpl	One Complement	Complementa a 1 el byte destino
dec	Decrement	Decrementa en 1 el valor del destino
halt	Halt	Detiene el oscilador
inc	Increment	Incrementa en 1 el valor del destino
iret	Interrupt routine return	Provoca un retorno al programa interrumpido
jp	Absolute Jump	Realiza un salto absoluto a la dirección destino
jra	Jump relative always	Realiza un salto relativo a la direción destino
jrt	Jump relative if true	Salto relativo si 1 (verdad)
jrf	Jump relative if false	Salto relativo si 0 (falso)
jrih	Jump relative if interrupt	Salto relativo si las interrupciones están a nivel alto
jril	Jump relative if interrupt low	Salto relativo si las interrupciones están a nivel bajo
jrh	Jump relative if half-carry	Salto relativo si el Flag H está a 1
jrnh	Jump relative if no half-carry	Salto relativo si el Flag H está a 0
jrm	Jump relative if interrupt mask	Salto relativo si el Flag I está a 1
jrnm	Jump relative if no interrupt mask	Salto relativo si el Flag I está a 0
jrmi	Jump relative if negative	Salto relativo si el Flag N está a 1
jrpl	Jump relative if positive or zero	Salto relativo si el Flag N está a 0
jreq	Jump relative if equal	Salto relativo si el Flag Z está a 1
jrne	Jump relative if not equal	Salto relativo si el Flag Z está a 0
jrc	Jump relative if carry	Salto relativo si el Flag C está a 1
jrnc	Jump relative if no carry	Salto relativo si el Flag C está a 0

TODAS las instrucciones ASSEMBLER para ST7

Neumó.	Descripción	Definición
jrult	Jump relative if lower than	Salto relativo si el Flag C está a 1
jruge	Jump relative if greater or equal	Salto relativo si el Flag C está a 0
jrugt	Jump relative if greater than	Salto relativo si los Flags C o Z están a 0
jrule	Jump relative if lower or equal	Salto relativo si los Flags C o Z están a 1
ld	Load	Carga el valor del fuente en el byte destino
mul	Multiply	Multiplica A por X o Y almacenando el resultado en (X o Y) + A
neg	Negate (2's complement)	Complementa a 2 el valor destino
nop	No operation	Ejecuta dos ciclos máquina
or	Logical Or	Realiza una operación OR entre A y el fuente, lo deja en A
pop	Pop from the stack or CC	Recupera desde la pila
push	Push into the stack	Salva en la pila
rcf	Reset carry flag	Pone a 0 el Flag C
ret	Return from subroutine	Recupera desde la pila el registro PC
rim	Reset interrupt mask	Pone a 0 el Flag I y habilita interrupciones
rlc	Rotate left throught carry	Rota un bit a la izquierda el valor destino con C
rrc	Rotate right throught carry	Rota un bit a la derecha el valor destino con C
rsp	Reset stack pointer	Pone a 0 el registro Puntero de pila
sbc	Subtraction with carry	Resta de A el valor del fuente y del Flag C
scf	Set carry flag	Pone a 1 el Flag C
sim	Set interrupt mask	Pone a 1 el Flag I y deshabilita interrupciones
sla	Shift left arithmetic	Desplaza un bit hacia la izquierda el valor destino
sll	Shift left logical	Desplaza un bit hacia la izquierda el valor destino
sra	Shift right arithmetic	Desplaza un bit a la derecha el valor destino
srl	Shift right logical	Desplaza un bit a la derecha el valor destino
sub	Substraction	Resta de A el valor del fuente
swap	Swap nibbles	Invierte los nibbles del byte destino
tnz	Test for negative or zero	Comprueba el byte destino activando los Flags N y Z
trap	Software interrupt	Fuerza una interrupción no enmascarable
wfi	Wait for interrupt	Pone la CPU a la espera de una interrupción
xor	Logical exclusive OR	Realiza una operación XOR entre A y el fuente, lo deja en A

NOTA: Como hemos expuesto en la introducción nuestra intención es proporcionar una tabla con todas las instrucciones Assembler para los microprocesadores ST7 en orden alfabético, si bien al llegar a las **instrucciones** de **salto condicional** (**JR**) hemos preferido seguir un orden lógico en detrimento del orden alfabético. Por esta razón hemos ido agrupando las instrucciones que afectan al mismo **Flag** del registro **Condition Code**.

ALGUNAS CUESTIONES antes de EMPEZAR

La presentación de las instrucciones es necesaria para entender la forma de escribirlas correctamente. Progresivamente vamos a ir presentando **tablas** con un nivel de detalle mayor para pasar del aspecto general a los detalles de las instrucciones.

A la **definición** de cada instrucción le acompaña su **sintaxis**, es decir la relación entre la instrucción y los operandos, y una **tabla sinóptica** que permite una consulta rápida sobre el **formato** de la instrucción y sobre los **Flags** afectados del registro **Condition Code** (**Código de Condición**).

También proporcionamos tablas detalladas con el **formato** de la instrucción (forma correcta de escribirla en relación a todos los posibles modos de direccionamiento), el **op-code** (codificación en formato ejecutable), los **ciclos maquina** (tiempo necesario para su ejecución) y su **longitud** en bytes. Por último proporcionamos una tabla con los **Flags** afectados del registro **Condition Code**.

Por supuesto que también hemos incluido varios **ejemplos** que ilustran el uso correcto de las instrucciones en cuestión.

1° GRUPO INSTRUCCIONES de CARGA (CLEAR y LOAD)

Las instrucciones **Clear** y **Load** tienen en común el hecho de que ambas **cargan** un valor en un registro o en una dirección de memoria. En efecto, **Clear** (limpiar) se puede interpretar como "**carga el valor 00**".

CLR (Clear)

Esta instrucción **carga** el valor **00** en el **byte** seleccionado como **destino**. La **sintaxis** de la instrucción es:

clr dst

Donde **dst** puede ser una dirección de memoria (**mem**) o un registro (**reg**).

Cuadro Sinóptico

neumo.	dst	H	I	N	Z	C
clr	mem			0	1	
clr	reg			0	1	

clr dst

dst	direccionamiento	op-code				ciclos	bytes
A	inherente		4F			3	1
X	inherente		5F			3	1
Y	inherente	90	5F			4	2
corto	corto, directo		3F	XX		5	2
(X)	X-indexado no-offset		7F			5	1
(corto,X)	corto, directo X-indexado		6F	XX		6	2
(Y)	Y-indexado no-offset	90	7F			6	2
(corto,Y)	corto, directo Y-indexado	90	6F	XX		7	3
[corto]	corto, indirecto	92	3F	XX		7	3
([corto],X)	corto, indirecto X-indexado	92	6F	XX		8	3
([corto],Y)	corto, indirecto Y-indexado	91	6F	XX		8	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	puesto a 0	puesto a 1	no influenciado

LD (Load)

Esta instrucción **carga** el valor del operando **fuente** (**src**) en el operando **destino** (**dst**). El valor del operando fuente queda inalterado. La **sintaxis** de la instrucción es:

ld dst,src

Donde **dst** y **src** pueden ser un **registro** (**reg**), una **dirección** de **memoria** (**mem**) o el byte menos significativo del **Puntero de Pila** (**Stack Pointer**, **S**).

Cuadro Sinóptico	neumo.	dst	src	H	I	N	Z	C
	ld	reg	mem			N	Z	
	ld	mem	reg			N	Z	
	ld	reg	reg					
	ld	S	reg					
	ld	reg	S					

ATENCIÓN: El formato **ld mem,mem** no está permitido, es decir no se puede cargar un valor de una dirección de memoria a otra. Para realizar esta operación primero hay que cargar el valor en un **registro** y luego desplazarlo a la **dirección** de **memoria**.

ld A,src

dst	src	direccionamiento	op-code				ciclos	bytes
A	#byte	immediato		A6	XX		2	2
A	corto	corto, directo		B6	XX		3	2
A	largo	largo , directo		C6	MS	LS	4	3
A	(X)	X-indexado no-offset		F6			3	1
A	(corto,X)	corto, directo X-indexado		E6	XX		4	2
A	(largo,X)	largo, directo X-indexado		D6	MS	LS	5	3
A	(Y)	Y-indexado no-offset	90	F6			4	2
A	(corto,Y)	corto, directo Y-indexado	90	E6	XX		5	3
A	(largo ,Y)	largo, directo Y-indexado	90	D6	MS	LS	6	4
A	[corto]	corto, indirecto	92	B6	XX		5	3
A	[largo.w]	largo, indirecto	92	C6	XX		6	3
A	([corto],X)	corto, indirecto X-indexado	92	E6	XX		6	3
A	([largo.w],X)	largo, indirecto X-indexado	92	D6	XX		7	3
A	([corto],Y)	corto, indirecto Y-indexado	91	E6	XX		6	3
A	([largo.w],Y)	largo, indirecto Y-indexado	91	D6	XX		7	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld dst,A

dst	src	direccionamiento	op-code				ciclos	bytes
corto	A	corto, directo		B7	XX		4	2
largo	A	largo , directo		C7	MS	LS	5	3
(X)	A	X-indexado no-offset		F7			4	1
(corto,X)	A	corto, directo X-indexado		E7	XX		5	2
(largo,X)	A	largo, directo X-indexado		D7	MS	LS	6	3
(Y)	A	Y-indexado no-offset	90	F7			5	2
(corto,Y)	A	corto, directo Y-indexado	90	E7	XX		6	3
(largo ,Y)	A	largo, directo Y-indexado	90	D7	MS	LS	7	4
[corto]	A	corto, indirecto	92	B7	XX		6	3
[largo.w]	A	largo, indirecto	92	C7	XX		7	3
([corto],X)	A	corto, indirecto X-indexado	92	E7	XX		7	3
([largo.w],X)	A	largo, indirecto X-indexado	92	D7	XX		8	3
([corto],Y)	A	corto, indirecto Y-indexado	91	E7	XX		7	3
([largo.w],Y)	A	largo, indirecto Y-indexado	91	D7	XX		8	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld X,src

dst	src	direccionamiento	op-code				ciclos	bytes
X	#byte	immediato		AE	XX		2	2
X	corto	corto, directo		BE	XX		3	2
X	largo	largo , directo		CE	MS	LS	4	3
X	(X)	X-indexado no-offset		FE			3	1
X	(corto,X)	corto, directo X-indexado		EE	XX		4	2
X	(largo,X)	largo, directo X-indexado		DE	MS	LS	5	3
X	[corto]	corto, indirecto	92	BE	XX		5	3
X	[largo.w]	largo, indirecto	92	CE	XX		6	3
X	([corto],X)	corto, indirecto X-indexado	92	EE	XX		6	3
X	([largo.w],X)	largo, indirecto X-indexado	92	DE	XX		7	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld dst,X

dst	src	direccionamiento	op-code				ciclos	bytes
corto	X	corto, directo		BF	XX		4	2
largo	X	largo , directo		CF	MS	LS	5	3
(X)	X	X-indexado no-offset		FF			4	1
(corto,X)	X	corto, directo X-indexado		EF	XX		5	2
(largo,X)	X	largo, directo X-indexado		DF	MS	LS	6	3
[corto]	X	corto, indirecto	92	BF	XX		6	3
[largo.w]	X	largo, indirecto	92	CF	XX		7	3
([corto],X)	X	corto, indirecto X-indexado	92	EF	XX		7	3
([largo.w],X)	X	largo, indirecto X-indexado	92	DF	XX		8	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld Y,src

dst	src	direccionamiento	op-code				ciclos	bytes
Y	#byte	immediato	90	AE	XX		3	3
Y	corto	corto, directo	90	BE	XX		4	3
Y	largo	largo , directo	90	CE	MS	LS	5	4
Y	(Y)	Y-indexado no-offset	90	FE			4	2
Y	(corto,Y)	corto, directo Y-indexado	90	EE	XX		5	3
Y	(largo ,Y)	largo, directo Y-indexado	90	DE	MS	LS	6	4
Y	[corto]	corto, indirecto	91	BE	XX		5	3
Y	[largo.w]	largo, indirecto	91	CE	XX		6	3
Y	([corto],Y)	corto, indirecto Y-indexado	91	EE	XX		6	3
Y	([largo.w],Y)	largo, indirecto Y-indexado	91	DE	XX		7	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld dst,Y

dst	src	direccionamiento	op-code				ciclos	bytes
corto	Y	corto, directo	90	BF	XX		5	3
largo	Y	largo , directo	90	CF	MS	LS	6	4
(Y)	Y	Y-indexado no-offset	90	FF			5	2
(corto,Y)	Y	corto, directo Y-indexado	90	EF	XX		6	3
(largo ,Y)	Y	largo, directo Y-indexado	90	DF	MS	LS	7	4
[corto]	Y	corto, indirecto	91	BF	XX		6	3
[largo.w]	Y	largo, indirecto	91	CF	XX		7	3
([corto],Y)	Y	corto, indirecto Y-indexado	91	EF	XX		7	3
([largo.w],Y)	Y	largo, indirecto Y-indexado	91	DF	XX		8	3

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	1 cuando el resultado es NEGATIVO	1 cuando el resultado es CERO	no influenciado

ld reg,reg ld reg,S ld S,reg

dst	src	direccionamiento	op-code				ciclos	bytes
X	A	inherente		97			2	1
A	X	inherente		9F			2	1
Y	A	inherente	90	97			3	2
A	Y	inherente	90	9F			3	2
Y	X	inherente	90	93			3	2
X	Y	inherente		93			2	1
A	S	inherente		9E			2	1
S	A	inherente		95			2	1
X	S	inherente		96			2	1
S	X	inherente		94			2	1
Y	S	inherente	90	96			3	2
S	Y	inherente	90	94			3	2

Condition Flags

H	I	N	Z	C
no influenciado	no influenciado	no influenciado	no influenciado	no influenciado

Nota: A continuación hemos preparado algunos ejemplos que os ayudarán a entender como utilizar las instrucciones **clear** y **load** y, sobre todo, a evitar algunos errores que se suelen cometer.

EJEMPLOS para el 1° GRUPO de INSTRUCCIONES

Es posible que algunos creáis que estas páginas son completamente superfluas ya que en artículos anteriores hemos visto numerosos ejemplos de uso de la instrucción **ld**.

Efectivamente, esta instrucción, por su naturaleza, es la que mejor se presta a ser utilizada en los ejemplos: Su uso es intuitivo y, sin necesitar muchas explicaciones, permite realizar ejemplos sencillos. Además **se utiliza bastante** en todos los programas y puede utilizarse con todos los modos de direccionamiento, a excepción de los modos **Bit Operation** y **Relativos**. Por este motivo la hemos utilizado cuando hemos explicado los modos de direccionamiento. Por tanto debería estar muy clara la forma de utilizarla, sin necesidad de analizar más ejemplos. Ahora bien, esta instrucción puede llevar a **provocar errores**, ya que a veces la instrucción tiene una apariencia correcta pero el compilador señala problemas.

Con los ejemplos, además de explicar la forma de utilizar las tablas, se **minimizará** el riesgo de **cometer errores** ocasionados por haber infravalorado su aparente sencillez.

Empezamos con la instrucción **clear**, analizando la siguiente instrucción de ejemplo:

clr x

Con esta instrucción ponemos a **0** el registro índice **X**, es decir forzamos a **00h** su contenido. Como se puede ver en la tabla correspondiente en la fase de compilación el código ejecutable de operación de esta instrucción (**op-code**) es **5F**. Después de su ejecución, el registro **X** contiene el valor **00h** y, por consiguiente, los Flags **N** y **Z** del registro **Condition Code** tienen los

dst	direccionamiento	op-code			
X	inherente		5F		

siguientes valores:

N=0 porque el **resultado** es **positivo**
Z=1 porque el **resultado** es igual a **cero**

NOTA: Los **Flags** del registro **Condition Code** (**Código de Condición**) fueron abordados en profundidad en la revista **N.229.**

Ahora vamos a analizar uno de los formatos más simples de la instrucción **load**:

ld a,#140

Con esta instrucción cargamos el valor numérico **140** en el registro **A**. Como se puede ver en la tabla correspondiente:

dst	src	direccionamiento	op-code			
A	#byte	immediato		A6	XX	

En la fase de compilación el código de esta instrucción (**op-code**) es:

A6 8C

Donde **8C** es el equivalente hexadecimal del valor decimal **140**.

NOTA: Al ser **8C** el valor del operando es el valor que reemplaza a **XX** en el op-code.

En la fase de ejecución en el registro **A** se carga el valor **140** (o **8Ch**). Después de su ejecución los Flags **N** y **Z** del registro **Condition Code** tienen los siguientes valores:
N=1 porque el **resultado** es **negativo**
Z=0 porque el **resultado** es distinto de **cero**

Veamos un nuevo ejemplo:

ld a,140

Con esta instrucción queremos cargar en el registro **A** el valor contenido en la **dirección** de **memoria 140** (**8Ch** en hexadecimal). También esta instrucción, además de ser formalmente correcta, origina un resultado coherente para el micro ST7LITE09. En efecto, la **dirección** de **memoria 8Ch** corresponde a una dirección de **Data RAM** del micro ST7LITE09 (ver Fig.1).

Como se puede ver en la tabla correspondiente:

dst	src	direccionamiento	op-code			
A	corto	directo corto		B6	XX	

En la fase de compilación se genera el siguiente **op-code**:
B6 8C
Supongamos, por ejemplo, que la dirección de memoria **8Ch** contiene el **valor 00h**. En fase de ejecución se cargará este valor en el registro **A**. Des-

pués de su ejecución los Flags **N** y **Z** del registro **Condition Code** tienen los siguientes valores:

N=0 porque el **resultado** es **positivo**
Z=1 porque el **resultado** es igual a **cero**

Ahora veamos que sucedería si, por error, escribiéramos:

ld a,1400

Con esta instrucción cargamos en el registro **A** el valor contenido en la dirección de memoria **1400** (**0578h** en hexadecimal). La instrucción es correcta en cuanto a su forma, como se puede ver en la tabla correspondiente:

dst	src	direccionamiento	op-code			
A	largo	directo largo		C6	MS	LS

El compilador **no** señala ningún **error** y genera el **op-code** siguiente:

C6 05 78

Desafortunadamente se trata de una instrucción **incorrecta** ya que en el caso del microcontrolador ST7LITE09 daría origen, si ejecutara, a un resultado imprevisible. En efecto, la dirección de memoria **0578h** corresponde a una dirección que **no** existe en el micro ST7LITE09 (el **mapa de memoria** con las direcciones válidas se muestra en la Fig.1). Como consecuencia en la fase de ejecución del programa, al final de esta instrucción, podríamos tener un valor imprevisto en el registro **A** y valores desconocidos y completamente arbitrarios en los Flags **N** y **Z**. Vamos a analizar ahora otra instrucción correcta solamente en apariencia. Nuestra intención es escribir una instrucción que cargue en el registro Índice **Y** el valor contenido en una posición específica de una **tabla** definida en **Program Memory** como se indica a continuación

(FB16) TABLA DC.B 22,95,24,96,108,3,90,1

En primer lugar hay que elegir la **posición** de la tabla que queremos utilizar. Por ejemplo, si queremos utilizar el **tercer byte** hay que escribir la siguiente instrucción:

ld x,#2

Ahora escribimos la instrucción con la que queremos cargar en el registro **Y** el valor presente en la dirección de memoria en la que está definida **TABLA** más el desplazamiento indicado por el registro **X** (**2**):

ld y,(TABLA,x)

A primera vista esta instrucción parece correcta, ya que utiliza el direccionamiento **Indexado Directo Largo**. Ahora bien, en la fase de compilación se generará el siguiente **error**:

"Error 54: Can't match addressing mode"

Es decir un error en el modo de direccionamiento. El error está provocado por el hecho de que cuando se utiliza un registro índice **Y** como **destino** también debe utilizarse como **fuente** o como **desplazamiento** (**offset**). Por tanto la instrucción correcta sería:

ld y,(TABLA,y)

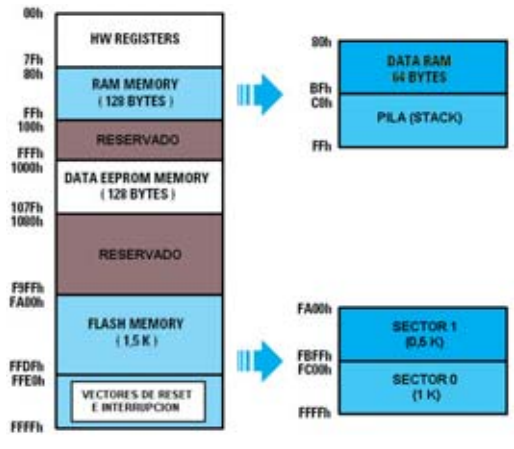

Fig.1 Mapa de memoria del micro ST7LITE09 dividido en áreas:

HW REGISTERS (ocupa de la dirección 00h a la dirección 7Fh).
RAM MEMORY (ocupa de la dirección 80h a la dirección FFh). Está dividida en dos zonas: Data RAM (de 80h a BFh) y Pila (de C0h a FFh).
DATA EEPROM MEMORY (ocupa de la dirección 1000h a la dirección 107Fh). Esta memoria mantiene los datos aunque el micro no esté alimentado.
FLASH MEMORY (ocupa de la dirección FA00h a la dirección FFDFh). Contiene las instrucciones del programa.
VECTORES de RESET e INTERRUPCIÓN (ocupa de la dirección FFE0h a la dirección FFFFh). Se utiliza para la gestión de interrupciones.

Que como se puede ver en la tabla correspondiente:

dst	src	direccionamiento	op-code			
Y	(largo.Y)	directo largo.Y.ind.	90	DE	MS	LS

Genera el siguiente código de operación (**op-code**):

90 DE FB 16

Donde **FB16h** (MS-LS) es precisamente la dirección de memoria en la que hemos definido la variable **TABLA**.

NOTA: Lo explicado para el registro **Y** también es válido para el registro **X**. Utilizando este registro habríamos tenido que escribir la instrucción:

ld x,(TABLA,x)

Que generaría el **op-code**:
DE FB 16

Donde **FB16h** (MS-LS) es precisamente la dirección de memoria en la que hemos definido la variable **TABLA**.

Llegado este punto, una vez ejecutada la instrucción, el valor inicial (**2**) contenido en el registro **Y** es reemplazado por el valor **24**, que es precisamente el valor contenido en el tercer byte de la tabla anteriormente definida. Después de su ejecución los **Flags** del registro **CC** tienen los siguientes valores:

N=0 porque el **resultado** es **positivo**
Z=0 porque el **resultado** es distinto de **cero**

Una solución alternativa, que comporta un número mayor de instrucciones, podría ser la siguiente:

ld x,#2
ld a,(TABLA,x)
ld y,a

En este caso primero cargamos en el registro **X** el valor numérico **2**, luego cargamos en el registro **A** el valor presente en el tercer byte de **TABLA**, es decir **24**, y por último se carga el valor contenido en **A** en el registro **Y**. Afrontamos ahora un último ejemplo en el que proponemos una instrucción correcta en la hemos empleado un registro particular, el registro **Puntero de Pila** (**Stack Pointer**, **S**). Este registro corresponde al **byte** menos significativo del registro **Stack Pointer** de **2 bytes** (**16 bits**).
NOTA: La **Pila** y el registro **Puntero de Pila** (**Stack Pointer**) fueron abordados en profundidad en la revista **N.233.**

Con la instrucción:

ld y,s

Cargamos en el registro **Y** el valor del **Stack Pointer**, "salvando" así la dirección actual de la **Pila** sin modificar el registro **Puntero de Pila** (**Stack Pointer**).

Que como se puede ver en la tabla correspondiente:

dst	src	direccionamiento	op-code			
Y	S	inherente	90	96		

Generaría el **op-code**:

90 96

En fase de ejecución, independientemente del valor contenido en el registro **Stack Pointer** almacenado en el registro **Y**, los **Flags** del registro **CC** no quedan afectados.

Esta instrucción "salva" el valor del **Stack Pointer** dejándolo **inalterado**, por lo tanto se puede utilizar en diferentes situaciones, por ejemplo cuando se tiene que realizar un "retorno rápido" de una serie de **subrutinas anidadas**, es decir una dentro de la otra, sin respetar el orden jerárquico. En este caso también se podría utilizar otra instrucción:

ld s,y

Con esta instrucción cargamos en el byte menos significativo del registro **Puntero de Pila** (**Stack Pointer LSB**) el valor anteriormente salvado en el registro **Y**. Después de esta instrucción el registro Stack Pointer "apuntará" a la nueva dirección de la **Pila**. Se trata de una instrucción "extremadamente peligrosa" ya que **modifica** el registro **Stack Pointer** y por lo tanto influye en las **direcciones de retorno** (instrucción **ret**) de las subrutinas llamadas por el programa o generadas por peticiones de interrupción. Resulta obvio que si no se utiliza con atención puede producir efectos desastrosos ya que modifica la propia ejecución del programa. Con esta instrucción los **Flags** del registro **CC no** quedan afectados.

CONCLUSIONES

Como habréis observado en las tablas de la instrucción **load** los únicos formatos que **no** afectan al contenido de los **Flags N** y **Z** son: **ld reg,reg** - **ld s,reg** - **ld reg,s**.