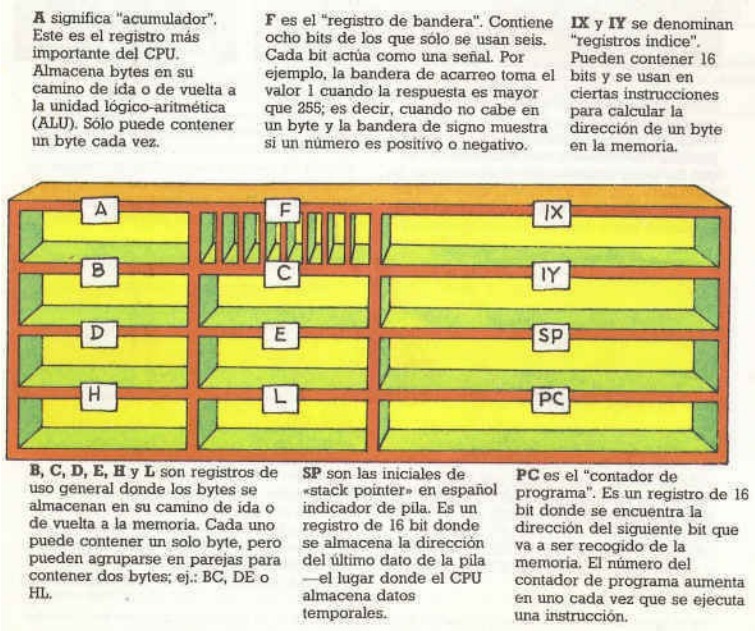
CAPÍTULO III. La Pantalla... y más

Lo que vemos en el monitor de nuestro CPC es reflejo del contenido de una zona de la memoria, llamada memoria de pantalla. Es por ello que antes de meternos de lleno en su conocimiento veamos cómo trabajan las direcciones y otros interesantes conceptos...



Sabemos ya cómo manejarnos con los registros de 8 bits, o sea de un byte; pero también sabemos que las direcciones que maneja el Z80 se componen de dos bytes (agrupación que se conoce como Word o Palabra). A fin de poder trabajar con direcciones, el Z80 nos permitirá usar los registros de 8 bits por parejas, esto es, BC, DE y HL, tal como se ven en la foto de nuestro Z80, con una utilización muy similar a los de 8 bits:

LD RR', NN	Carga con un número de 16 bits ese par.
LD RR', (NN)	Carga desde una dirección (similar a PEEK).
LD (NN), RR'	Carga en una dirección (similar a POKE).

Fijémonos que en las instrucciones de carga hacia/desde la dirección (NN) lo que estamos desplazando son **dos bytes**, correspondiendo uno a cada registro "simple" (recordemos que en cada posición de memoria sólo cabe un byte). Estos dos bytes estarán ubicados uno en la dirección indicada por (NN) y el otro en la siguiente; si examinamos esas posiciones de memoria veremos que se han colocado el byte de bajo orden en NN y el de alto orden en NN+1. Esto es debido al funcionamiento interno del Z80 y aunque nuestro programa ensamblador nos evita preocuparnos de esto, nos será útil tenerlo en cuenta.

También tenemos disponibles las útiles instrucciones INC y DEC para esos pares de registros:

INC RR' y DEC RR'

Incrementa/decrementa en 1 el número en RR'.

Pero el par HL es bastante especial, de hecho es el niño mimado de nuestro Z80, ya que se le están permitidas algunas instrucciones que no pueden hacer otros registros, incluso algunas que tenía en exclusividad el Acumulador, por ejemplo las sumas y restas:

ADD HL,RR ADC HL,RR y SBC HL,RR

HL admite que se le sume (ADD), sume con acarreo (ADC) o reste con acarreo (SBC) cualquiera de los otros pares de registros o sí mismo, o sea, que están permitidas las instrucciones ADD HL,DE, ADC HL,HL o SBC HL,BC.

Estas instrucciones INC y DEC no afectan al registro de banderas, la ADD sólo afecta a la de Carry y SBC y ADC a todas.

Aparte de estos pares de registros, el Z80 tiene también una serie de registros de 16 bits que son los que tenemos a la derecha en la foto: los índices **IX** e **IY**, el contador de programa **PC** y el puntero de la pila **SP**. A alguno de ellos ya lo vimos en la introducción: hablo del **PC** o Contador de Programa que recordemos que guarda la dirección de la SIGUIENTE instrucción a ser ejecutada. Pero antes de empezar a contaros de estos registros tengo que hablaros de

La Pila (o Stack), que es una parte de la memoria que el Z80 va a utilizar como almacén de direcciones: va a ir metiendo allí las direcciones que le interese recordar, de el siguiente modo: cuando hacemos un CALL pone la dirección del CALL en el **PC** pasando la que contenía éste a la pila (esto es, la SIGUIENTE a la instrucción CALL), y cuando la rutina llega a su RET, recuperará la dirección que guardó en la pila colocándola en el PC de nuevo para seguir con el programa.

La pila recibe su nombre de la manera en que se almacenan los datos, que es similar a una pila (de libros, por ejemplo). Voy apilando los elementos uno sobre otro, de manera que si quiero retirar alguno de la pila, tendré que sacar el último que he colocado, después el penúltimo, y así.

El **SP** (Stack Pointer) o Puntero de la Pila contiene *la dirección* donde se guarda el último dato introducido en la pila. Al arrancar el **CPC** la pila comienza en la dirección &BFFE (justo delante de la memoria de pantalla) y va creciendo **hacia abajo**, de

manera que al introducirse una dirección en la pila, el SP desciende dos unidades y al sacar una dirección el SP aumenta dos unidades.

El ejemplo que más representa el funcionamiento de la pila es que es como un clavo que sobresale del techo en el que vamos clavando papeles (las direcciones). Va creciendo hacia abajo y siempre tenemos que sacar el último que hemos metido.

Nosotros también podremos meter y sacar datos de la pila, normalmente para proteger el contenido de algún registro ante la llamada de una rutina del firmware (¿recordais el ejemplo del Capítulo 1 que nos corrompía los registros?). Para ello tenemos las instrucciones

PUSH RR' Mete el número de 16 bits contenido en RR y resta dos al SP.
POP RR' Saca un número de 16 bits de la pila y suma dos al Stack Pointer.

Eso sí, no debemos olvidarnos de que los CALL y RET también usan la pila. Si dentro de un CALL metemos un número en la pila y no lo sacamos antes de que llegue su RET, el programa no RETornará a donde debía y los resultados pueden ser desastrosos.

PUSH y POP nos permiten, además de todos los pares de registros, meter también en la pila el par ficticio **AF**, por si queremos salvaguardar el registro de Flag, y los registros de 16 bits **IX** e **IY.**

(HL) Hasta aquí, estas instrucciones de 16 bits funcionan *básicamente* de la misma manera que las de ocho bits, pero la posibilidad de usar paréntesis con los pares de registros nos abre nuevas posibilidades. Recordemos que igual que (NN) quería decir *el contenido de la dirección NN,* **(HL)** quiere decir *el contenido de la dirección apuntada por el registro doble HL*.

El Z80 va a tratar (HL) como si fuera un registro de 8 bits más, estando permitidas **INC (HL) y DEC (HL), ADD A,(HL) y SUB(HL), LD (HL),N , LD (HL),R y LD R,(HL)**…

Una primera aplicación para esto puede ser el uso de este registro doble como puntero, por ejemplo, **LD A,(HL)**, e ir avanzando o retrocediendo posiciones de memoria mediante INC HL o DEC HL. Esto se conoce como indexación de una porción de memoria.

Un pequeño ejemplo, que además nos enseña como mostrar un texto en pantalla:

```
ORG &4000          ; Dirección de Inicio, instrucción para el Ensamblador.
LD A,(longitud)    ; Cargamos B con el contenido de .longitud, es el número
LD B,A             ; de caracteres del texto a escribir.
LD HL,data         ; Carga HL con la dirección de .data  donde está el texto.
.escribe
LD A,(HL)          ; Carga A con el contenido de la dirección apuntada por
                   ; HL, que es el siguiente caracter a escribir.
CALL &BB5A         ; Rutina del Firmware TXT OUTPUT, imprime en
                   ; pantalla el caracter cuyo ASCII está en el Acumulador.
INC HL             ; INCrementa HL, apuntando al siguiente caracter.
DJNZ escribe       ; Decrementa B y si no hay Zero, salta a escribe.
RET                ; RETorna a donde estaba.
.longitud          ; Aquí guardamos la longitud del texto a escribir.
DB 14              ; Directiva del ensamblador Define Byte.
.data              ; Aquí guardamos el texto a escribir.
DB "Hola, que tal?" ;Directiva del ensamblador Define Byte.
```

Los ***DB*** que aparecen en ***.data*** y ***.longitud*** son directivas del Ensamblador, las cuales ya mencionamos en el Capítulo I. Ésta en particular, ***Define Byte,*** introduce los bytes que le indiquemos a partir de esa posición de memoria. En nuestro ejemplo, en ***.data*** tendríamos que haber puesto los ASCII correspondientes a "H", "o", "l",... y así sucesivamente, en esas posiciones de memoria, pero al utilizar las comillas ya indicamos al Ensamblador que lo que queremos es meter el ASCII de esos caracteres.

En otros listados, esta directiva puede venir como **DEFB, BYTE, DEFM** o **TEXT** pero Maxam nos entenderá cualquiera de ellas. Otras directivas del ensamblador son ***DS*** o ***DEFS, Define Storage*** que nos reserva un espacio en memoria del número de bytes que le indiquemos y ***DW*** o ***DEFW, Define Word*** que nos almacena direcciones de memoria (o números de 16 bits), guardando primero el byte de bajo orden y después el de alto, como indicábamos más arriba.

Los registros índice **IX** e **IY** comparten muchas de las instrucciones de carga y suma con HL como se ve en el cuadro resumen del final del Capítulo, así como instrucciones que hacen un uso parecido de los paréntesis al que hace HL, indexando la dirección contenida en dicho registro más un desplazamiento positivo o negativo como el que usábamos en los saltos relativos, permitiendo por tanto indexar 127 posiciones por delante y 128 por detrás de dicha dirección, que se mantiene en dicho registro IX o IY como referencia o índice, de ahí su nombre. Los operandos se indican **(IX + d)** e **(IY + d)** y admiten practicamente las mismas instrucciones que (HL) considerándose registros de 8 bits también. Por tanto, en las instrucciones de Capítulos anteriores donde dijimos **R** podemos incluir también **(HL)**, **(IX+d)** e **(IY+d)**.

Sin embargo, en los otros dos pares de registros **BC** y **DE** el uso con paréntesis está limitado a cargas hacia/desde el Acumulador (**LD A,(BC) ó LD (BC),A**).

Y ahora ya sí que estamos preparados para ver **La Pantalla...**

La memoria de pantalla va de &C000 a &FFFF ocupando por tanto 16 Kb, siendo &C000 la esquina superior izquierda y &FFFF la inferior derecha (esto no es rigurosamente cierto, como veremos en Capítulos posteriores).

En la pantalla hay 25 renglones de caracteres de ocho píxeles de alto (los caracteres tienen 8x8 píxeles), y se almacenan en memoria en primer lugar todas las primeras líneas de cada renglón de caracteres, luego las segundas... y así hasta ocho, que son los píxeles de altura de los caracteres. Dentro de un carácter cada píxel está separado verticalmente del siguiente &800 y, cada caracter del siguiente, &50. De la última fila a la primera siguiente hay 30 bytes que no se muestran.

Por tanto, en todos los modos tenemos 200 píxeles de alto (25 renglones de caracteres x 8 píxeles de alto cada caracter). Horizontalmente, cada línea tiene 80 bytes (&50), pero el número de píxeles horizontales dependerá del modo en que nos encontremos:

* En modo 0 hay 160 píxeles de ancho, teniendo dos píxeles por byte (2x80 bytes de ancho de pantalla) y 16 pinceles (PEN) seleccionados con los bits 1-5-3-7 para el píxel izquierdo y 0-4-2-6 para el derecho.

* En modo 1 hay 320 píxeles de ancho, teniendo cuatro píxeles por byte (4x80) y 4 pinceles (PEN) seleccionados con los bits 3-7 para el primer píxel, 2-6 para el segundo, 1-5 para el tercero y 0-4 para el cuarto.

* En modo 2 hay 640 píxeles de ancho, con 8 píxeles por byte (8x80) y 2 pinceles (PEN) seleccionados con los bits de 7 a 0 para los ocho píxeles de izquierda a derecha.

&C000	&C001	&C002	&C003	&C004	...
&C800	&C801	&C802	&C803	&C804	...
&D000	&D001	&D002	&D003	&D004	...
&D800	&D801	&D802	&D803	&D804	...
&E000	&E001	&E002	&E003	&E004	...
&E800	&E801	&E802	&E803	&E804	...
&F000	&F001	&F002	&F003	&F004	...
&F800	&F801	&F802	&F803	&F804	...
&C050	&C051	&C052	&C053	&C054	...
&C850	&C851	&C852	&C853	&C854	...
...	...	...	...	...	...

Espacio ocupado en memoria por el caracter de la posicion superior izquierda de pantalla en Modo 0.

Los pinceles tendrán los colores que les hayamos asignado de los 27 disponibles en nuestro CPC.

Vamos a ver el funcionamiento de una conocida rutina que nos dibujará un gráfico almacenado en memoria, del mismo modo que estaba guardado el texto del ejemplo anterior:

```
imp_grafico     ; Dibuja en pantalla el gráfico.
                ; Entradas:
                ;   BC Alto - Ancho (en bytes).
                ;   DE Origen (gráfico)
                ;   HL Destino (pantalla)
                ; Se alteran HL, BC, DE, AF.

.loop_alto
  PUSH BC       ; Guarda el alto y el ancho en la pila.
  LD B,C        ; Mete el ancho en B.
  PUSH HL       ; Guarda la dirección de destino en la pila.
.loop_ancho     ; Dibuja una línea del gráfico.
  LD A,(DE)     ; Carga A con un byte del gráfico.
  LD (HL),A     ; Dibuja en la posición de pantalla el byte en A.
  INC DE        ; Incrementa DE (apunta al siguiente byte).
  INC HL        ; Incrementa HL (apunta a la siguiente posición a dibujar).
  DJNZ loop_ancho ; Vuelve a dibujar otro byte hasta acabar con el ancho.
  POP HL        ; Recupera la dirección inicial de la línea de destino.
  LD A,H        ; Carga el A con H...
  ADD &08       ; ...para poder sumarle 8...
  LD H,A        ; ...y pasar a la siguiente línea  (añade 800 a HL).
  SUB &C0       ; Comprueba si se ha acabado el caracter, si al sumar 8...
  JP NC,sig_linea ; ...se ha rebasado FF.
  LD BC,&C050   ; Si se rebasa FF añade &C050 a la dirección de destino
  ADD HL,BC     ; y sigue con la primera línea del siguiente caracter.
.sig_linea
  POP BC        ; Recupera el Alto-Ancho de la pila...
  DJNZ loop_alto ; ...y le resta la línea recién dibujada.

RET
```

¿Te has fijado en que no he puesto el ORG al principio? Esto es porque ésta es una rutina para incluir en un programa y que además necesita una serie de condiciones de entrada antes de ser ejecutada y que son las que tenemos indicadas al principio junto con las de salida.

Hacer rutinas con este formato es muy interesante, ya que nos permite reutilizarlas para otros programas que hagamos después. Tras varios programas nos podemos juntar con una buena librería de rutinas que nos ahorrarán mucho trabajo.

Por ejemplo, la rutina de escribir texto de más arriba quedaría:

```
escribe_texto   ; Escribe un texto en la posición actual del cursor.
                ; Entradas
                ;    HL contiene la dirección del texto precedido por el
                ;    número total de caracteres del texto.
                ; A, B y HL son modificados.
LD B,(HL)       ; Carga en B el número de caracteres del texto...
INC HL          ; ... y pasa a apuntar al primer carácter.

.escribe
  LD A,(HL)     ; Carga A con el contenido de la dirección apuntada por
                ; HL, que es el siguiente carácter a escribir.
  CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                ; pantalla el caracter cuyo ASCII está en el Acumulador.
  INC HL        ; INCrementa HL, apuntando al siguiente caracter.
  DJNZ escribe  ; Decrementa B y si no hay Zero, salta a escribe.

RET             ; RETorna a donde estaba.
```

Volviendo con lo que estábamos, para diseñar nuestro gráfico hemos echado mano a Sprot para PC, de ESP Soft y al modelo que teníamos más a mano... (ejem !) Para CPC existen también programas como Graphic City de Ubi Soft, Sprite Definer de Sean McManus o Panda Sprites de Interceptor Software, pero no me he puesto a fondo con ellos, ya que Sprot se adecua sobradamente a nuestro propósito.

El siguiente programa dibuja nuestro gráfico en la pantalla y lo desplaza por ella con las teclas O, P, Q, A de una manera que todos conocemos :-). Recordemos que antes de ensamblar debemos copiar también la rutina **imp_grafico** de más arriba:

```
ORG &4000

LD A,&0          ; La rutina del Firmware SCR SET INK (&BC32)
LD B,&0          ; establece los colores contenidos en B y C para el número
LD C,&0          ; de pincel contenido en A. B y C han de ser iguales para
CALL &BC32       ; un color, y distintos para alternar entre esos colores.
                 ; AF, BC, DE y HL se corrompen.
LD A,&1          ; Lo mismo para el pincel 1.
LD B,&5
LD C,&5
CALL &BC32

LD A,&2          ; Para el pincel 2.
LD B,&1A
LD C,&1A
CALL &BC32

LD A,&3          ; Y para el pincel 3.
LD B,&6
LD C,&6
CALL &BC32

LD A,1
CALL &BC0E       ; Ponemos la pantalla en Mode 1. (SCR SET MODE)

LD HL,&C2F6      ; Posición inicial del gráfico en pantalla, centradito.


.dibuja
PUSH HL          ; Guardamos la nueva posición en la pila.
LD HL,(ultima)   ; Cargamos la última posición del gráfico.
LD BC,&1707      ; Longitud y ancho.
LD DE,borra      ; Dirección donde está almacenado el gráfico de borrar.
CALL imp_grafico ; Dibuja el gráfico de borrar.
POP HL           ; Recupera la nueva posición de la pila.
PUSH HL          ; Guardamos la nueva posición en la pila.
LD BC,&1707      ; Longitud y ancho.
LD DE,grafico    ; Dirección donde está almacenado el gráfico.
CALL imp_grafico ; Dibuja el gráfico.
```

```
.teclado          ; Recoge en A una tecla.
CALL &BB1B
CP "q"            ; Salta a donde corresponda según se haya pulsado o, p,
JP Z,arriba       ; q o a. Si es x sale del programa.
CP "a"
JP Z,abajo
CP "o"
JP Z,izquierda
CP "p"
JP Z,derecha
CP "x"
RET Z
JP teclado

.arriba
POP HL
LD (ultima),HL    ; Guarda en .ultima la posición actual.
LD BC,&50
SBC HL,BC         ; Resta &50 a la posición actual.
LD A,H            ; Vemos si nos salimos de la pantalla por arriba.
SUB &C0           ; Si el contenido de HL es menor de &C000, nos hemos
JP NC,dibuja      ; salido de la zona ocupada por la memoria de pantalla.
ADD HL,BC         ; Vuelve a dejar HL como estaba y lo devuelve a la pila.
PUSH HL
JP teclado        ; Vuelve a esperar una tecla.

.abajo
POP HL
LD (ultima),HL    ; Guarda en .ultima la posición actual.
LD BC,&50
ADD HL,BC         ; Añade &50 a la posición actual.
LD BC,&C730       ; Comprueba que el gráfico no rebase la línea entre
SBC HL,BC         ; &C730 y &C77F y se salga de la pantalla por la parte
JP C,vuelve1      ; inferior de la pantalla.
ADC HL,BC
LD BC,&C77F
```

```
SBC HL,BC
JP NC,vuelve2
ADC HL,BC
LD BC,&50
SBC HL,BC
PUSH HL
JP teclado
. vuelve1
ADD HL,BC
jp dibuja
.vuelve2
ADD HL,BC
jp dibuja


.izquierda
POP HL
LD (ultima),HL      ; Guarda en .ultima la posición actual.
DEC HL
LD A,H              ; Vemos si nos salimos de la pantalla por arriba
SUB &C0
JP NC,dibuja
INC HL
PUSH HL
JP teclado


.derecha
POP HL
LD (ultima),HL      ; Guarda en .ultima la posición actual.
INC HL
LD BC,&C72A         ; Vemos si nos salimos de la pantalla por abajo. Hemos
SBC HL,BC           ; elegido &C72A porque es la última posición de pantalla
JP C,vuelve         ; en que se muestra nuestro gráfico entero.
ADD HL,BC
DEC HL
PUSH HL
JP teclado
```

```
.vuelve
ADD HL,BC
jp dibuja


ultima DW &C2F6


.grafico
db &00,&01,&00,&00,&00,&00,&00
db &03,&00,&2A,&8B,&00,&00,&00
db &13,&00,&7F,&8C,&00,&02,&00
db &11,&88,&FF,&CC,&00,&03,&00
db &30,&FD,&55,&CC,&00,&EE,&00
db &70,&FF,&FF,&FC,&FF,&FE,&00
db &70,&FF,&09,&55,&FF,&F8,&00
db &70,&F7,&88,&77,&FE,&F0,&80
db &F0,&F7,&88,&77,&FE,&F0,&80
db &F0,&F3,&8D,&FF,&FC,&F0,&80
db &F0,&F3,&FF,&77,&F8,&F0,&80
db &F0,&F1,&CC,&FF,&F8,&F0,&80
db &F0,&F1,&FF,&FF,&F0,&F0,&00
db &F0,&F1,&FF,&FE,&F0,&F0,&00
db &70,&71,&FF,&FE,&F0,&B0,&00
db &60,&77,&FF,&FC,&E0,&30,&00
db &40,&FF,&FF,&88,&C0,&10,&00
db &40,&EE,&77,&CC,&40,&10,&00
db &22,&CC,&11,&EE,&00,&00,&00
db &11,&CC,&00,&66,&44,&00,&00
db &00,&EE,&00,&77,&CC,&00,&00
db &00,&00,&00,&33,&00,&00,&00
db &00,&00,&00,&66,&00,&00,&00
```

```
.borra
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
db &00,&00,&00,&00,&00,&00,&00
```

Repasamos:

LD RR', NN ; Carga con un número de 16 bits el par de registros.
LD RR', (NN) ; Carga desde una dirección (similar a PEEK).
LD (NN),RR' ; Carga en una dirección (similar a POKE).

Intrucciones válidas para BC, DE, HL, IX e IY.

INC RR' ; Incrementa en 1 el número en BC, DE, HL, IX o IY.
DEC RR' ; Decrementa en 1 el número en BC, DE, HL, IX o IY.

ADD RR',RR' ; Añade a HL, IX ó IY el número contenido en BC, DE o sí mismo.
ADC HL,RR' ; Añade con acarreo a HL el número contenido el par RR'.
SBC HL,RR' ; Resta con acarreo a HL el número contenido en ese par.

PUSH RR' ; Mete en la pila el número de 16 bits contenido en RR' y resta dos al Puntero de la pila (SP).
POP RR' ; Saca un número de 16 bits de la pila, lo mete en RR' y suma dos al puntero de la pila (SP).

Siendo RR' AF, BC, DE, HL, IX o IY.

(HL), (IX+d) e (IY+d) equivalen a cualquier registro R de 8 bits en las instrucciones de los Capítulos anteriores.
LD A,(BC) ó (DE) ; Carga el Acumulador con el contenido de la dirección apuntada por BC o DE.
LD (BC),A ó (DE) ; Carga la posición de memoria apuntada por BC o DE con el contenido del Acumulador.