AMSTRAD
CPC-HARDWARE

Conocimiento y ampliacion

ALAN TREVENNOR

ra-ma

AMSTRAD CPC-HARDWARE

ALAN TREVENNOR

ra-ma

Título de la obra original:

UNDERSTANDING and EXPANDING YOUR
AMSTRAD CPC-464, 664 and 6128.
por Alan Trevennor.

Publicado en Gran Bretaña por:

SIGMA PRESS
98A Water Lane
Wilmslow, Cheshire, Reino Unido

Traducción del inglés por:
M.C. Dopazo.

Editado por:

RA-MA Editorial
Carretera de Canillas, 144
28043 MADRID (España)

ISBN: 84-86381-19-3
Dep. Legal: M. 35953 - 1986
Signo Impresores, S.A.
Albasanz, 27-28037 Madrid

PCB y Servicio de Kits

Se encuentran disponibles los kits y los materiales de todos los proyectos de este libro, incluyendo tarjetas de circuito impreso de alta calidad y diseñadas profesionalmente. Esto incluye la tarjeta maestra de ocho conectores de expansión del 'bus' y la fuente de alimentación - como se detalla en el apéndice cuatro y en el capítulo tres.

El único suministrador de estos 'kits', circuitos impresos especiales y proyectos terminados es:

Halstead designs,
5 High St,
Halstead,
Essex,
United Kingdom.

Teléfono (0787) 477408

Servicio de Software

El juego completo del software de este libro está disponible en disco solamente en:

Readers software service
Amsoft
Brentwood house
Brentwood
Essex CM145YA

Teléfono (0277) 230222

El disco que se suministra contiene varios programas que no están listados ni descritos en este libro, junto con varios ficheros de texto.

Nota para los Usuarios de CPC 664 y 6128

Este libro se comenzó a escribir cuando solo estaba disponible el CPC 464. Debido a que la cantidad de desarrollo que ha llevado este libro, tomó un tiempo considerable su terminación. Mientras tanto, Amstrad lanzó el CPC 664 y el 6128. Es una medida de consistencia por parte de Amstrad el mantener la compatibilidad de los nuevos productos con el CPC 464 en la mayoría de los aspectos. Esto significa que casi todo el material contenido en este libro se puede aplicar a las nuevas máquinas: los usuarios de 664 y 6128 pueden considerar que los proyectos detallados dentro de este libro serán compatibles en su mayor parte. Hay casos donde la aplicación de algún programa (como el catálogo por impresora del Capítulo 5) no serán aplicables para las máquinas que no tengan cassette, pero estos programas están marcados como para los usuarios de CPC 464 solamente.

La descripción de los datos sobre los discos se han evitado deliberadamente ya que hay otro libro de Sigma, "The Amstrad disc companion" por Simon Williams, que explica el subsistema de discos de Amstrad y el software de soporte.

Introducción

Este libro ha sido escrito con el ánimo de darle a los lectores información en dos áreas. Primero, se proporciona información para una buena comprensión del 'hardware' que hay dentro de la familia Amstrad CPC de ordenadores personales. (No se detallan el 'hardware' ni el 'sofware' del disco porque hay otro libro de Sigma, "The Amstrad disc companion" por Simon Williams, que lo hace). Segundo, algunos proyectos prácticos que han sido desarrollados especificamente como periféricos del CPC. Junto con estos proyectos, proporcionamos el software que los maneja, y detalles completos de los programas escritos especialmente para ellos. Los proyectos no solo amplían el campo de sus actividades con ordenadores, sino que algunos de ellos se han diseñado para hacer al ordenador más rápido y más sencillo de manejar y para que proporcione aún más facilidades.

Los lectores que tienen concentrada su atención en los programas, son bienvenidos al maravilloso mundo del 'hardware', le aseguramos que dentro de este libro hay suficiente información como para iniciarle en el mundo de los chips, así como presentarle suficiente información sobre los programas para mantener a los chips bajo control.

La mayor parte de los problemas de la construcción de 'hardware' se han evitado con los diseños de algunas tarjetas de circuito impreso (PCB). Estos los ha diseñado Halstead Designs Ltd que está preparando también kits de los componentes y versiones ya montadas de los proyectos. Amsoft va a suministrar el juego completo de programas por medio de su servicio de 'software' para usuarios. (En la página iii de este libro puede encontrar la información completa de estos servicios). La disponibilidad del 'hardware' y del 'software' hace más fácil el proceso de construcción de cualquiera de los proyectos descritos.

No recomiendo a los lectores ningún orden de lectura en particular, excepto que los que tienen pocos conocimientos sobre 'hardware' de ordenadores, lean primero el apéndice seis. Este apéndice es una introducción a algunos de los conceptos básicos que se usan a lo largo de este libro.

Quiero agradecer a la gente que ha contribuido de una u otra forma a la terminación de este libro. A mucha gente de Amsoft Brentwood de cuyos cerebros he tomado un poco, especialmente a William Poel por su cooperación. A Geof Melvin y Wally Maynard - mis jefes en Systime que me asistieron en diversas forma. A Graham Beech de Sigma por su ánimo, y por su paciencia. Gracias a todos.

Pero mi mayor agradecimiento debe ir a mi esposa, Wendy, que ha pasado muchas tardes sola, con la única compañía del click distante del teclado, y los lánguidos gritos de desesperación, para recordarle que todavía tiene un marido! Gracias cariño.

Dedicado a Wendy con amor.

Alan Trevennor Diciembre, 1985.

Contenido

CAPITULO UNO
Revisión del 'Hardware'

El CPC consta de dos partes principales. Una es el monitor, que puede ser de dos tipos diferentes, monocromo o de color. En ambos casos el monitor contiene, además del tubo de rayos catódicos cuya parte visible es la pantalla, la fuente de alimentación para la otra parte.

La segunda parte es la unidad del teclado. Esta contiene la parte inteligente de la máquina, el microprocesador Z80A. Dentro del teclado están también la memoria (capaz de contener 65536 octetos de datos - o si usted prefiere, 65536 caracteres alfabéticos), los chips que permiten la comunicación con el teclado, el chip que realiza los efectos de sonido y la música, y el grabador de cassette junto con su parte electrónica. Además contiene un chip propio, único en el CPC, que reduce drásticamente el número de chips requeridos. Está también el chip que controla la pantalla, y finalmente, la ROM (Read Only Memory - Memoria de Solo Lectura) que contiene el sistema operativo y el intérprete BASIC, que controla todo el 'hardware' y le permite programar el ordenador en lenguaje BASIC.

En este primer capítulo echaremos un vistazo a todos estos chips y veremos cómo se usan en el CPC, y también algunas de sus características de operación. No es mi propósito dar una descripción exhaustiva de cada chip, ya que no tendría un valor práctico. Solamente la información que pueda ser útil para añadir 'hardware' a la máquina, y para comprender las funciones 'hardware' que se incluyen.

LOS CHIPS DENTRO DEL CPC

El Z80A

Este es el chip microprocesador que sirve de procesador central a toda la máquina. Esto es todo lo que se necesita saber en este momento, ya que el siguiente capítulo está enteramente dedicado a él.

El PPI 8255

El 8255 fue diseñado por Intel Corporation. Se le llama Programable Peripherial Interface (PPI) (Interfase Periférico Programable). Está contenido en un chip de 40 patas DIL (Dual In Line), y es un chip muy versátil para su uso en Entrada y Salida del ordenador (E/S). El 8255 proporciona 24 líneas de interfase entre el ordenador y el mundo exterior, formando un canal por el que se pueden pasar datos hacia y desde el ordenador. En efecto, el PPI proporciona hasta tres canales de ocho bits, de entrada o salida. A estos canales se les llama puerta A, puerta B y puerta C.

Cuando se enciende el ordenador, el PPI se pone en estado restaurado (RESET). En este estado el 'software' puede introducir instrucciones en el registro de definición de modo del PPI, que definirá sus características de operación. Hay muchas maneras de configurarlo. Se puede configurar con tres puertas de salida, tres puertas de entrada, o cualquier mezcla intermedia. Adicionalmente hay modos especiales

que permiten que ciertos bits de la puerta C se usen como señales de entrada para retener datos en las puertas A o B, o como vía para seguir el protocolo de señales de trasferencia de datos, así como de señales relacionadas con las interrupciones. Como ya hemos dicho, las funciones exactas del chip las define el programa que lo inicializa al conectarlo. Para obtener más información de las configuraciones disponibles refiérase a la hoja de caracteristicas del 8255. Veamos algunas cosas sobre el PPI.

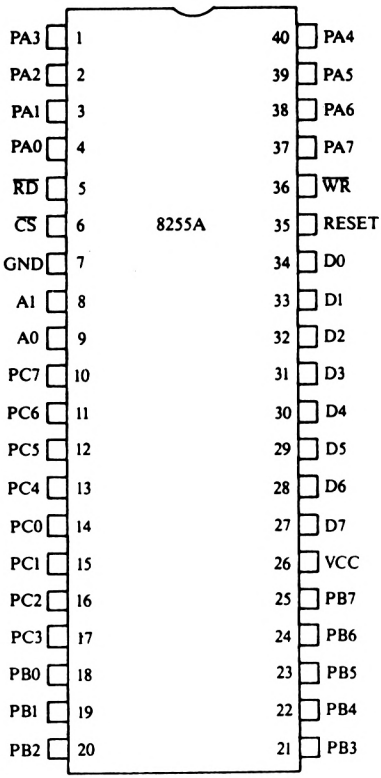


Fig 1.1:Detalle de las conexiones del chip 8255A

En la figura 1.1 podemos ver las conexiones del 8255. Requiere solamente +5 voltios de alimentación y una conexión de masa (0 voltios). Cada una de las tres puertas A, B y C, tiene 8 bits numerados como PA0:PA1:PA2:PA3:.. para la puerta A, PB0:PB1:PB2:PB3:.. para la puerta B, y así sucesivamente. El 8255 ocupa cuatro direcciones del espacio de E/S del CPC464. La primera dirección es para la puerta A, la segunda para la puerta B, y la tercera para la puerta C. Cualquie-

ra de estas tres direcciones puede ser leída o se puede escribir en ella. La cuarta dirección solo admite la escritura, no se puede leer de ella, esta es la dirección donde se escribe la definición del modo de funcionamiento. La tabla de la verdad de direccionamiento de este chip se muestra en la figura 1.2.

A1	AO	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Función
0	0	0	1	0	Leer puerta A
0	1	0	1	0	Leer puerta B
1	0	0	1	0	Leer puerta C
0	0	1	0	0	Escribir en puerta A
0	1	1	0	0	Escribir en puerta B
1	0	1	0	0	Escribir en puerta C
1	1	1	0	0	Escribir octeto de control
X	X	X	X	1	Chip deseleccionado

X = no tiene importancia su estado

Nota: Si se lee el registro de control, obtendremos siempre basura, ya que
no existe la facilidad de leerlo. Las puertas A, B y C se pueden leer aún cuando se hayan programado como de salida.

Fig 1.2:Tabla de direccionamiento del 8255A

El 8255 es una pieza muy importante en la lógica de E/S, ya que se puede programar para cubrir los requerimientos específicos de casi todas las aplicaciones que necesiten entrada/salida.

Ahora echemos una ojeada al uso del 8255 dentro del ordenador CPC464. Los registros del 8255 residen en las direcciones hexadecima-

les de E/S F4xx, F5xx, F6xx y F7xx. La parte xx de estos números significa que se puede usar cualquier número para la segunda parte de la dirección. Esto se debe al hecho de que las direcciones de E/S en el CPC464 no están completamente decodificadas - ya hablaremos más de ello en la sección final de este capítulo. Las direcciones anteriores corresponden a las puertas A, B y C y al registro de control del 8255, respectivamente.

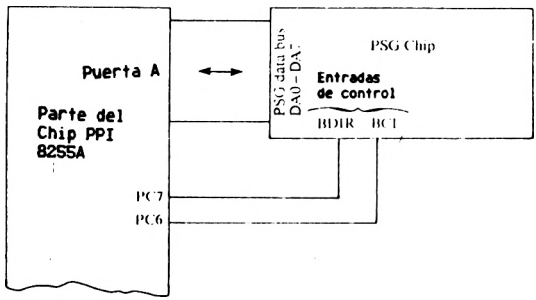


Tabla de verdad de Entradas de cont 1

BDIR	BC1	Transaction
1	1	PPI Port A to PSG Register select
1	0	PPI Port A to the selected PSG register
0	1	The contents of the selected PSG Reg to Port A
0	0	Inactive

Fig 1.3:Diagrama de bloques que muestra la conexión del chip PSG al 8255A

```
1000  CLS: PRINT "Este programa introduce en memoria un programa en":
      PRINT "código: máquina, comenzando en la dirección 30000.":
      PRINT " Este pequeño programa parece que deja la máquina"
1010  PRINT "colgada hasta que se pulsa la tecla de CAPS LOCK en el":
      PRINT "teclado. Vea también el listado en ensamblador de este":
      PRINT "programa, y lea los comentarios"
1020  MEMORY 25000: RESTORE: FOR I=30000 TO 30099: READ R: POKE I,R: NEXT:
      PRINT STRING$(3,10): INPUT "Pulse INTRO para continuar"; R$:
      CLS: PRINT "Ahora llamamos a la rutina en código máquina - pulse "
1030  PRINT "CAPS LOCK cuando haya comprobado que pulsar cualquier otra":
      PRINT "tecla no produce ningún efecto": CALL 30000
30000 DATA 197 , 245 , 243 , 62 , 130 , 205 , 133 , 117 , 62 , 0
30010 DATA 205 , 127 , 117 , 62 , 7 , 205 , 139 , 117 , 62 , 192
30020 DATA 205 , 127 , 117 , 62 , 0 , 205 , 127 , 117 , 205 , 139
30030 DATA 117 , 62 , 128 , 205 , 127 , 117 , 62 , 0 , 205 , 127
30040 DATA 117 , 62 , 14 , 205 , 139 , 117 , 62 , 192 , 205 , 127
30050 DATA 117 , 62 , 0 , 205 , 127 , 117 , 62 , 146 , 205 , 133
30060 DATA 117 , 62 , 72 , 205 , 127 , 117 , 1 , 0 , 244 , 237
30070 DATA 120 , 203 , 119 , 32 , 250 , 241 , 193 , 251 , 201 , 1
30080 DATA 0 , 246 , 237 , 121 , 201 , 1 , 0 , 247 , 237 , 121
30090 DATA 201 , 1 , 0 , 244 , 237 , 121 , 201 , 0 , 0 , 0
```

Fig 1.4:Listado del programa de interrogar por CAPS LOCK

Entonces, ¿para qué se usan las tres puertas?. La puerta A se usa para acceder al chip PSG (Generador Programable de Sonido). Como toda la transferencia de información desde y hacia el PSG tiene lugar por

la puerta A, debe estar cambiando constantemente de puerta de entrada a puerta de salida, y viceversa. La cuestión se complica aún más por el echo de que el PSG por sí mismo ya tiene una puerta de E/S, que se usa para ayudar en el manejo del teclado. Por todo esto el sistema operativo no solo se tiene que ocupar en tratar el flujo de datos entre la puerta A y el PSG, sino que tiene que usar dos bits de la puerta C para introducir datos de control en el PSG, códigos que definen el tipo de transacción requerida. Todo esto puede inducir a confusión al principio, por lo que incluimos la figura 1.3 para ilustrar la idea con más claridad. ¿Todavía no lo ve claro?. Vamos a ver, las figuras 1.4 y 1.5 son los listados de dos programas similares que cargan los registros del PPI y del PSG con los valores requeridos para comprobar si se ha pulsado la tecla CAPS LOCK. Ya sabemos que se puede usar la función **INKEY** del BASIC para hacer lo mismo, pero nosotros lo vamos a hacer de una manera más difícil - ¡solamente por esta vez!. La figura 1.4 es la versión BASIC del programa, que incluye el código máquina dentro de las sentencias DATA. La figura 1.5 es el listado en ensamblador de este programa. No se asuste si no entiende el programa en este momento, vamos a introducirle en el código máquina en el capítulo 5, e incluiremos la forma en que puede incorporar sus programas en código máquina dentro de sus programas BASIC. El programa termina cuando se pulsa la tecla de CAPS LOCK. Como puede ver en el listado del programa, el manejo del teclado no es de fácil acceso al programador, pero usando la puerta de E/S del PSG se evita tener que poner otro chip para leer el teclado. Puede usar el programa para leer cualquier tecla del teclado, solo tiene que ver el diagrama del teclado en la descripción del uso de la puerta C.

```
100 ;** Programa para mostrar cómo se accede al teclado por medio de los
200 ;** chips PPI y PSG.
300 ;
400         ORG  30000
500         ENT  $
600         PUSH BC ; Salvar registros
700         PUSH AF
800         DI
900         LD   A,#82
1000        CALL CONOUT ; Poner el modo del PPI
1100        LD   A,0    ; Puerta C
1200        CALL COUT   ; manda cero para inactivar el PSG
1300        LD   A,7    ; Después puerta A = 7
1400        CALL AOUT
1500        LD   A,#C0  ; manda C0 a puerta C como una
1600        CALL COUT   ; dirección de registro PSG
1700        LD   A,#0   ; e inactiva PSG
1800        CALL COUT
1900        CALL AOUT   ; Puerta A también a cero
2000        LD   A,#80  ; escribe cero en el registro 7 del PSG
2100        CALL COUT   ; pone comando escribir en puerta C
2200        LD   A,0
2300        CALL COUT   ; Inactiva de nuevo la puerta C
2400        LD   A,14   ; Ponemos el número de registro E/S de PSG
2500        CALL AOUT   ; en el PSG por medio de la puerta A
2600        LD   A,#C0
2700        CALL COUT   ; mandamos comando bloqueo dirección a puerta C
2800        LD   A,0    ; y volvemos a inactivar la puerta C
2900        CALL COUT
3000        LD   A,#92  ; Ahora debemos cambiar el modo del PPI
3100 ; esto implica el cambio de la puerta A como entrada.
3200        CALL CONOUT
```

```
3300         LD   A,#48  ; dile a la puerta C que prepare el PSG para mandar
3400 ; su registro 14, selecciona el bloque en que reside CAPS LOCK
3500         CALL COUT
3600         LD   BC,#F400  ; BC=dirección de puerta A
3700 NOPRES: IN   A,(C)
3800         BIT  6,A
3900         JR   NZ,NOPRES ; saltar si no está pulsada
4000         POP  AF
4100         POP  BC
4200         EI          ; activa de nuevo las interrupciones
4300         RET
4400 COUT:   LD   BC,#F600 ; Dirección puerta C a BC
4500         OUT  (C),A
4600         RET
4700 CONOUT: LD   BC,#F700 ; BC=dirección registro control de PPI
4800         OUT  (C),A
4900         RET
5000 AOUT:   LD   BC,#F400 ; Dirección puerta A a BC
5100         OUT  (C),A
5200         RET
```

Fig 1.5: Versión en ensamblador del programa de interrogar CAPS LOCK

La puerta B del PPI se usa como entrada de 8 bits. El bit 0 detecta el impulso de sincronismo del chip controlador del VDU (Unidad de Vídeo o pantalla). Los bits 1 a 4 se usan para comprobar si están instalados los puentes de la placa principal del CPC. La función de la mayoría de estos puentes no está definida en la documentación publicada por Amstrad, sin embargo, el puente 4 (LK4) que va conectado al bit 4 de la puerta B, no debe estar instalado si está conectado a una fuente de alimentación de 50 Ciclos, y debe estar instalado si lo tiene conectado a una fuente de 60 Ciclos (como en USA). Si teclea el comando BASIC:

```
PRINT BIN$(INP(&F500))
```

El BASIC imprimirá el estado de cada bit de la puerta B. En mi máquina obtengo 00111110. Cuando se lee como uno un bit controlado por uno de estos puentes, quiere decir que está desconectado. Por lo tanto, como en mi máquina los bits 1 a 4 son todos uno, todos los puentes están desconectados. El bit 5 de la puerta B es la entrada de una línea que está conectada al bus de expansión. Por lo tanto es razonable que se le llame EXP. Se puede usar para permitir a los programas comprobar si hay algo conectado en el bus de expansión. El bit 6 es la señal de BUSY (ocupado) de la puerta de la impresora Centronics. Si esta señal está alta (uno lógico) la impresora está indicando al ordenador que no le mande más caracteres para imprimir. (Vea el apéndice 7 para una explicación más detallada de la puerta de impresora). El bit 7 de la puerta B se usa para recibir datos desde los circuitos de recuperación de datos del cassette. La señal que aparece normalmente en este bit es la versión digital de los datos que están entrando desde la cinta. El sistema operativo tiene que decodificar esta señal, usando varias rutinas complejas de temporización, para obtener datos válidos. Todas las entradas de la puerta B, excepto los bits 0 y 7, tienen una resistencia de 2.2K para mantenerlas a uno lógico.

La puerta C usa todos los bits como salida. Los bits 0 a 3 van a un chip 74LS145. El código que aparece en estos bits es decodificado por

el 74LS145 para obtener diez salidas. Esto quiere decir que las diez salidas del 74LS145 se activan dependiendo del código en la entrada.

Las salidas del 74LS145, junto con los registros de E/S del PSG, forman una matriz X,Y entre cuyas conexiones se conectan las teclas del teclado. La figura 1.6 nos muestra dónde se conecta cada tecla dentro de la matriz. Cuando se pulsa una tecla, una de las salidas del 74LS145 se conecta momentáneamente con una de las entradas de la puerta de E/S del PSG. Como tenemos diez salidas del 74LS145 y ocho entradas al PSG, podemos tener 80 teclas conectadas a la matriz. Hay nueve intersecciones sin usar en el teclado QWERTY. Estas son las que se conectan cuando se usan las palancas de juego en la puerta que hay en la parte trasera del CPC 464.

El bit 4 se usa para controlar el relé del motor del cassette. Si se pone este bit a uno, el motor se pone en marcha por medio de un pequeño relé situado dentro del reproductor de cassette. Pruébelo por usted mismo tecleando el programa de la figura 1.7. Lo que hace es encender y apagar el motor cada dos segundos. NO INTENTE hacerlo con más frecuencia, ya que puede dañar el transistor que llama al relé controlado por el bit 4. El bit 5 de la puerta C se usa para mandar datos al cassette. Los bits 6 y 7 se usan para controlar las transacciones entre el PSG y la puerta A - como ya hemos visto anteriormente en la descripción del uso de la puerta A.

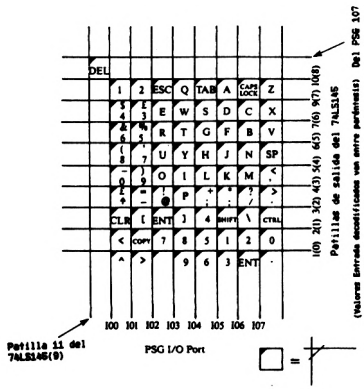


Fig 1.6:Diagrama de la matriz del teclado del CPC

```
1000  '** Programa para demostrar cómo controla la puerta C del PPI
      ** el relé del motor del cassette.

1100  ON BREAK GOSUB 10000
1200  EVERY 100,0 GOSUB 5000: MODE 2: LOCATE 1,10
1300  GOTO 1300
5000  OUT &F600,((INP(&F600) XOR 16)):
      PRINT "Cambiado el estado del relé";CHR$(24);CHR$(13);: RETURN
10000 OUT &F600,((INP(&F600) AND &E0)): PEN #0,1: PAPER #0,0: CLS:
      PRINT CHR$(13);"Relé desconectado": END '** Se asegura de que cuando
      ** pulsa ESC se desconecta el motor y la pantalla se vuelve normal.
```

Fig 1.7:Demostración del control del relé

La ROM

El intérprete de BASIC y el sistema operativo están programados permanentemente en una ROM (Memoria de solo lectura) de 32K octetos. Cada una de estas piezas vitales de 'software' ocupan exactamente la mitad de la ROM (esto es 16K octetos cada una) con el intérprete de BASIC en la mitad superior. Aunque residen físicamente en el mismo chip, la ROM está dividida en dos mitades. A lo largo de la documentación de Amstrad, y en este libro, hay referencias a la parte baja y alta de la ROM. El decodificador de dirección incluido en el grupo de puertas lógicas, controla todo esto de forma que la mitad inferior de la ROM comienza en la dirección 0000, mientras que la mitad superior comienza en la dirección hex C000. El diagrama de la memoria se complica aún más por el hecho de que la ROM se puede desconectar, y dejar la RAM en su lugar. Al final de este capítulo hay una sección especial que explica como funciona esta dualidad. El chip de la ROM por sí mismo tiene 15 entradas de direccionamiento, 8 líneas de E/S de datos, un patilla para activar la salida, una de selección del chip, y requiere solamente +5V y 0V. Su diagrama de conexiones es el mismo de la EPROM 27128 que podrá encontrar en la descripción del proyecto ocho.

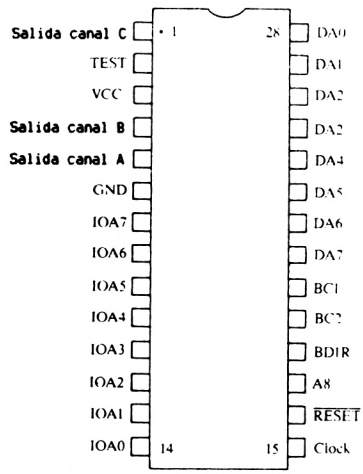


Fig 1.8: Diagrama de conexiones del chip PSG

Los Chips de RAM

La RAM consta de ocho chips. Son todos de 64K por un bit de RAM dinámica, del tipo HM4864. Aunque tienen la ventaja de ofrecer mayor capacidad por chip que la RAM estática (por ejemplo la RAM 6116 que tiene 2K por ocho bits, que se ve mucho en proyectos de revistas), la RAM dinámica tiene el inconveniente de que se necesita estar diciendoles continuamente lo que deben recordar. Esto se debe a que los elementos de memoria son realmente pequeños condensadores, que pierden su carga si no se les carga continuamente. Esta carga constante se llama refrescarlas(refresh). El truco está en realizar el refresco de la memoria cuando no es accedida. En efecto, el Z80A tiene la facilidad de realizar el refresco automáticamente, pero en el CPC se realiza con el grupo de puertas lógicas. Se hace así porque el

controlador de VDU también accede a la memoria antes de que pueda ocurrir el refresco. El proceso de refresco de memoria consiste en dos líneas pulsantes de entrada conectadas a todos los chips de memoria RAM. Se les llama CAS (Column Address strobe - Sincronismo de dirección de columna) y RAS (Row Address strobe - Sincronismo de dirección de fila). Son pulsados mientras recorren las direcciones de filas y columnas de la RAM secuencialmente. Los chips 4864 que se usan tienen un ciclo de 200 nanosegundos.

Potencialmente la RAM ocupa todo el mapa de la memoria, pero como ya dijimos en la sección anterior de este capítulo, la compleja decodificación de direcciones del CPC puede intercambiar secciones de la RAM para dejar paso a secciones de la ROM.

El Generador Programable de Sonido

El Generador Programable de Sonido (PSG) se usa de una manera realmente extraña en el CPC. El AY-3-8912 fue diseñado originalmente para ser conectado directamente al bus de un microprocesador. Como ya vimos cuando hablamos de la forma en que se usa el chip PPI 8255, el PSG no está conectado al bus del Z80A, recibe todos los datos y los comandos por medio del PPI. Esto no sobrecarga el chip, pero ciertamente sobrecarga a los programadores que quieren acceder directamente al PSG. No puedo imaginar que que haya mucha gente que, habiendo usado los comandos SOUND, ENV y ENT en el BASIC de Locomotive Software, quieran programar el PSG en código máquina. ¡El uso del PSG desde el BASIC ya es suficientemente complejo! Sin embargo, un examen detallado del PSG y sus registros internos nos puede revelar cómo consigue el BASIC mantener a la bestia bajo control.

En la figura 1.8 se muestra el diagrama de conexiones del chip PSG. Requiere +5 y 0 voltios. Las líneas DA0-DA7 son líneas bidireccionales tri-estado y están previstas para ser conectadas al bus de datos de un microprocesador. Las líneas marcadas como IOA0-IOA7 son una puerta de E/S de uso general - muy similares a la puerta de E/S del PPI. Esta puerta de E/S es la que vimos al tratar la forma en que el PPI y el PSG realizan la detección del teclado. Las patillas 1, 4 y 5 son de las que sale el sonido producido por el PSG. Como ya sabrá, si ha usado el comando SOUND del BASIC, usted puede definir cual de los tres canales producirá el sonido independientemente de los otros. El Locomotive BASIC tiene rutinas realmente ingeniosas para sincronizar la salida de estos tres canales. La patilla de RESET del PSG, cuando se pone a cero, pone en condición inactiva todos los registros internos del PSG. La entrada A8 no se usa en el CPC. La entrada TEST no suele usarse. La entrada CLOCK se conecta a una señal de un megaciclo de los circuitos internos del CPC. Si vuelve a la figura 1.3 de este capítulo, cómo se codifica la condición de las patillas BC1 y BDIR.

Veamos ahora los registros internos del PSG. El BASIC hace funcionar el el PSG por medio de una lista de parámetros suministrados por el usuario en el comando SOUND. Como en:

```
10 FOR I = 35 TO 4 STEP -1: SOUND 1,I,5: SOUND 2,I*4,5: SOUND 4,I*3,5:
   NEXT: GOTO 10
```

Un método similar se usa cuando se programa el PSG en código máquina. La diferencia más importante es que los parámetros que se usan deben ser colocados dentro de cada registro del PSG en el momento apropiado. Incluimos aquí un listado de los registros del PSG y su función.

Los tres canales de sonido dentro del PSG se llaman A, B y C. La frecuencia del sonido que sale de cada uno de estos canales se define en seis registros, dos por canal.

Registro 0 = Un registro de 8 bit que representa los ocho bits inferiores
de la frecuencia que se producirá en el canal A

Registro 1 = Un registro de 4 bits que representa los 4 bits superiores de
la frecuencia que se producirá en el canal A

Registro 2 = Igual que el 0 pero para el canal B

Registro 3 = Igual que el 1 pero para el canal B

Registro 4 = Igual que el 0 pero para el canal C

Registro 5 = Igual que el 1 pero para el canal C

Los valores colocados en los registros 0-5 se usan para definir el factor de división del reloj maestro. Por lo tanto, las frecuencias obtenidas dependerán de la frecuencia del reloj maestro. El registro 6 define la longitud del ruido. Este ruido es útil para las explosiones de los juegos, etc. R6 es un registro de 5 bits. R7 es el que activa la puerta de E/S y el mezclador de los registros de control. Los bits 0, 1 y 2, cuando están a cero, activan el generador de ruido para la salida de los canales A, B y C respectivamente. El bit 6 se pondrá a uno para hacer que la puerta de E/S sea de salida y a cero para hacerla de entrada. El bit 7 del registro 7 no se usa. Los registros 8, 9 y 10 son los controles de amplitud de los canales A, B y C respectivamente. Los bits 0 a 3 de estos registros controlan el volumen de cada canal, y el bit 4 controla la amplitud. Los registros 11, 12 y 13 se usan para controlar la envolvente de los sonidos generados. Los registros 11 y 12 se usan con un valor de 16 bits que define la longitud de la función seleccionada en el registro 13. Las funciones que se pueden seleccionar en el registro 13 son:

Detenido (Bit 0)
Alternado (Bit 1)
Ataque (Bit 2)
Continuar (Bit 3)

El resto de los bits de R13 no se usan. Finalmente, como ya hemos mencionado anteriormente, el registro 14 es el registro de E/S - usado exclusivamente como registro de entrada en el CPC, para la detección del teclado QWERTY.

El PSG es un chip bastante complejo. Para dominarlo debe pasar una gran cantidad de tiempo experimentando con las envolventes y los extras que suministra el Locomotive BASIC.

El Grupo de Puertas Lógicas

Este es un chip que ha sido diseñado exclusivamente para esta máquina, y ya lo hemos mencionado al principio de este capítulo. Su desarrollo evita la inclusión de muchos otros chips, que se requerirían para construir una configuración equivalente. No es fácil conseguir detalles sobre este chip, ya que su desarrollo seguramente habrá costado una gran cantidad de dinero. Por su posición en el circuito

se pueden adivinar la mayoría de las funciones que realiza. Como ya hemos mencionado, el grupo de puertas lógicas realiza el refresco de memoria. También hemos visto que debe tener un decodificador programable de direcciones, para que se puedan intercambiar la ROM y la RAM dentro del espacio de direcciones. (Vea la última sección de este capítulo si quiere más detalles del mapa de memoria). Otra cosa muy importante que produce el grupo de puertas lógicas es las señales RGB (Rojo Verde Azul), LUM y SYNC que van a la sección del monitor del CPC464 para proporcionar la información a la unidad de la pantalla y producir los gráficos y el texto que aparecen en ella.

Otra función del grupo de puertas lógicas es la generación de interrupciones. Todo el tiempo que el CPC está conectado, el grupo de puertas lógicas está produciendo interrupciones al microprocesador, las rutinas de interrupción hacen cosas como actualizar el reloj que cuenta el tiempo que ha pasado desde que se encendió el CPC, y lo usa la variable TIME del BASIC, y también para la revisión del teclado, que se realiza todo el tiempo que la máquina está encendida - a menos que usted active expresamente las interrupciones con un programa en ensamblador.

El grupo de puertas lógicas procesa también el reloj maestro de toda la máquina - convirtiendo los impulsos del cristal de 16 MHz en una señal de 4 MHz que se usa como señal de reloj para el Z80A. (No piense que el Z80A está trabajando a 4 MHz, porque la lógica interna del grupo de puertas lógicas expande la duración de las señales de requerimiento de memoria y de E/S reduciendo la frecuencia al equivalente a 3,5 MHz). El grupo de puertas lógicas produce señales de control para abrir y cerrar los bloques de memoria intermedia entre la RAM y el bus de datos del Z80A. También controla si las direcciones de RAM se direccionan por el VDU o la CPU.

El Chip Controlador del VDU

La parte electrónica de la pantalla consiste en un monitor monocromo o en color, con una fuente de alimentación integrada. Las únicas conexiones entre la unidad del teclado y el monitor son la fuente de +5 voltios, que proporciona la alimentación necesaria a la unidad del teclado, y las señales del monitor, mandadas al monitor desde el chip controlador de VDU que se encuentra dentro de la unidad del teclado.

En conjunción con el chip del grupo de puertas lógicas, el controlador VDU produce todas las señales necesarias para crear la imagen que usted ve en su monitor. Como ya sabrá esta puede ser texto, gráficos animados o cualquier cosa que pueda hacer su programa.

El chip controlador VDU es un MC6845. Como puede ver en la hoja de características reproducida en el apéndice 3, el 6845 se diseñó originalmente para trabajar con la familia de microprocesadores Motorola 6800, pero se puede adaptar para usarlo con otros sistemas. El 6845 puede compartir un área de la memoria principal con el microprocesador, y crear una imagen en el monitor a partir del contenido de la memoria. En su aplicación en el CPC, las puertas lógicas de vídeo controlan los colores que se muestran, mientras que el 6845 controla la forma y la posición de lo que se está mostrando. Como todas las imágenes de televisión, la pantalla del CPC está formada por puntos. Cada punto es la unidad direccionable más pequeña en la pantalla. Puede comprobarlo mediante la siguiente cadena de comandos BASIC:

```
MODE 2: FOR I=1 TO 500: POKE (&C000+(INT(RND*&4000)),128: NEXT
```

Esto construye en la pantalla una alfombra de estrellas como las que se usan en muchos juegos de guerras del espacio. Cada punto iluminado en la pantalla corresponde a un punto de la imagen. Lo que realmente hacen estas instrucciones es poner a uno el octavo bit de una posición aleatoria de la memoria de la pantalla, que comienza en la posición hexadecimal C000, para hacer que brille el punto. Ejecute la secuencia de comandos un par de veces más, pero sustituyendo el MODE por cero y uno. Notara que los puntos no son mayores en dirección vertical, sin embargo son más anchos en modo cero que en los modos uno y dos. Esto se debe a la forma en que el 6845 organiza la memoria de imagen.

Cuando se usa la pantalla para mostrar texto en la pantalla, hay un juego de caracteres dentro de la ROM que contiene una lista de las máscaras de puntos que se requieren para construir cada carácter, los puede imaginar como una lista de puntos que deben brillar para construir una letra, símbolo o número determinado. Cuando se quiere mostrar un carácter en la pantalla, se localiza en la lista la máscara de puntos y se coloca en la memoria de la pantalla en la dirección apropiada. El sistema operativo del CPC le permite también definir caracteres propios en una tabla en RAM. Así usted puede redefinir los caracteres que se muestran para unos códigos ASCII determinados. A esta facilidad se accede mediante la palabra SYMBOL del Locomotive BASIC.

El 6845 está leyendo continuamente el contenido de la memoria de la pantalla y mandándolo al flujo serie del grupo de puertas lógicas, que lo modifica añadiéndole información de color de su memoria interna de paleta. El grupo de puertas lógicas lo manda después al monitor para mostrarlo. Todo el proceso se sincroniza de forma que un punto determinado aparecerá siempre en el mismo punto de la pantalla en cada ciclo, dando la ilusión de una imagen estática. Cuando usa colores parpadeantes, los código adecuados de las tintas que usted ha elegido se colocan alternativamente en la paleta en memoria, en los intervalos requeridos para hacer parpadear la posición. Esto se realiza como parte de las rutinas de interrupción que hemos mencionado anteriormente.

¿Qué pasa con el cursor? Cuando está editando, trabajando con un proceso de texto o introduciendo programas, necesita tener una indicación visual de dónde va a aparecer el siguiente carácter que usted teclee. Para esto es para lo que está el cursor. El cursor se genera realmente en el 6845. Puede ser grande o pequeño, puede parpadear o estar estático, y también puede desaparecer. El 6845 usa varios registros para controlar el cursor y sus atributos, refiérase a la hoja de características para más detalles.

Hay otra facilidad del 6845 que es de especial utilidad para aquellos que quieren realizar gráficos interactivos, es la entrada del lápiz luminoso. Esta entrada es una patilla del 6845 que se extiende al bus de expansión (ver apéndice 5), Para usarlo debe conseguir un sensor de luz, instalarlo dentro de un bolígrafo viejo, del que se haya quitado previamente la punta y el tubo de tinta, y conectarlo mediante un circuito simple de interfase a la entrada del lápiz luminoso. Como cada pasada del punto de dibujo construye la imagen en la superficie del monitor, la luz proveniente del cátodo del tubo ilumina el sensor de luz momentáneamente cuando se mantiene la punta del lápiz sobre la pantalla.

El 6845 tiene varios contadores internos que le dicen exactamente que posición de memoria se está trasfiriendo a la pantalla. Asociados con la entrada del lápiz luminoso hay dos registros de ocho bits. Estos se cargan con el contenido de los contadores de la posición de memoria de pantalla cuando aparece una señal alta en la entrada del lápiz luminoso. Esto significa que una vez que ha construido el lápiz luminoso, que es bastante fácil y barato, solo necesita un programa que lea continuamente el estado de los registros de dirección del lápiz luminoso, y escriba un uno en dentro de la posición de memoria que le indica. Después puede mover su lápiz alrededor de la pantalla y dibujar fácilmente sobre ella.

El 6845 es un chip muy versátil y capaz. Cuando se usa en el CPC junto con el grupo de puertas lógicas, el resultado es una unidad de pantalla muy flexible, y se pueden hacer gráficos muy complejos. Los usos de la pantalla en color del CPC solamente están limitados por su imaginación.

La Puerta de la Impresora

En términos puros de 'hardware' la puerta de impresora consiste en un chip de anclaje (latch) de ocho bits. Este es el 74LS273, y está conectado a la dirección de E/S hex EF00. Hay algunas consideraciones especiales que se deben aplicar cuando se usa la puerta de la impresora, y las impresoras conectadas a ella. Todo esto se trata en el apéndice 7.

EL MAPA DE MEMORIA (Y COMO ALTERARLO)

Como ya hemos mencionado varias veces, el mapa de memoria del CPC es algo difícil debido al decodificador programable que hay en el grupo de puertas lógicas. Si lo intenta desde el BASIC puede hacer PEEK y POKE desde cualquier posición de memoria, y siempre parecerá que es RAM. La lógica nos dice que esto no puede ser así, debe haber alguna ROM en algún lugar, que contenga el intérprete BASIC y el sistema operativo. Algún dispositivo que cambie la ROM y la RAM cuando intentamos hacer PEEK y POKE de las posiciones de memoria. La realidad de todo esto es que el grupo de puertas lógicas tiene un registro en su interior que se llama el registro de selección de ROM. Ahora vamos a ver cómo funciona.

ATENCION: Si no sabe nada acerca del microprocesador Z80, lea primero el capítulo dos y vuelva después a leer esta sección.

En el capítulo dos veremos en detalle el microprocesador Z80A. Este microprocesador nos proporciona dos espacios de direccionamiento que son, el espacio de direcciones de memoria y el espacio de direcciones de E/S. En el esquema de direcciones del CPC el espacio de E/S se usa para dispositivos periféricos de E/S, y también para acceder al grupo de puertas lógicas. En un espacio de direcciones de E/S un acceso a una dirección específica siempre leerá o escribirá desde o hacia el mismo dispositivo o registro. El espacio de direcciones de memoria se divide en tres áreas distintas:

1) El área de la ROM inferior - direcciones 0000 a 03FF

2) El área media de la RAM - direcciones 4000 a BFFF

3) El área de la ROM superior - direcciones C000 a FFFF

El registro de selección de la ROM, que se carga escribiendo en la dirección de E/S 7F80, puede hacer que la ROM superior o inferior sea reemplazada por RAM. Esto se debe la decodificador programable de direcciones que hay dentro del grupo de puertas lógicas. Cuando el registro de selección muestra que, por ejemplo, la ROM inferior está desactivada, el decodificador activará la RAM cuando se decodifique una dirección del rango C000 a FFFF. Si está activada la ROM superior, el decodificador activa la ROM para que se pueda leer de ella, y la RAM permanece sin seleccionarse, excepto la parte media. La ROM inferior se puede conectar y desconectar de la misma forma. El esquema se complica más por el echo de que no importa qué ROM está activa, al escribir en una posición siempre se escribe en la RAM. Cuando está activada una ROM se dice que, la RAM a la que usted va a escribir, está "solapada" con la ROM.

Recuerde que aunque hemos hablado de la ROM superior e inferior, las dos residen realmente en dos mitades del mismo chip. La mitad a la que se accede se controla mediante el grupo de puertas lógicas.

EL DATACORDER (SOLO EN EL CPC 464)

Dentro del CPC464 hay un mecanismo estándar de cassette y algo de electrónica conectada a él. Coloque una cinta que contenga música en el Datacorder, pulse la tecla PLAY y teclee el comando CAT. Gire el control de volumen que hay al lado de la unidad. ¿No es exactamente HI-FI, verdad? Intente poner una cinta que contenga palabras, suena bastante mal, como si la oyera por un receptor de radio muy malo. La razón para la mala reproducción es que la electrónica que procesa la señal de la cabeza del Datacorder está preparada para optimizar solamente la frecuencia en la que se graban los datos, no el amplio rango de frecuencias que se necesitan para reproducir música normal. (Quite ahora la cinta de música - no queremos escribir sobre ella).

Cuando escribe datos a la cinta, el sistema operativo del CPC tiene un control limitado sobre el movimiento de la cinta, mediante el relé de parada/arranque, cuyos contactos conectan o desconectan una fuente de +5 voltios al motor del cassette. La electrónica del Datacorder tiene una señal de entrada y otra de salida - como un grabador de audio. Los datos que se escriben a la cinta pasan en forma serializada a la puerta de entrada desde una línea de la puerta C del PPI. Los datos leídos desde la cinta se presentan como formas de onda cuadradas en la línea de salida. La electrónica del Datacorder no contiene circuitos lógicos, toda la codificación y decodificación, serialización y deserialización de los datos se hace por programa.

LAS UNIDADES DE DISCO

No se dan detalles extensos sobre las unidades de disco, ya que hay otro libro de SIGMA (The Amstrad Disk Companion de Simon Williams) que se concentra en la versión de disco del CPC464 así como en el CPC664 y el 6128.

Las unidades de disco que se usan en las máquinas CPC son unidades de disco flexible de tres pulgadas. Los discos son de dos caras y

proporcionan 169K de almacenamiento por cada cara del disco. En el CPC464 el controlador se proporciona en un conector externo que contiene no solamente el chip controlador de disco flexible, sino también una ROM con la parte del sistema operativo correspondiente a los comandos extra que controlan la E/S de disco. Por si esto no fuera suficiente, el paquete contiene una copia del sistema operativo CP/M, y el lenguaje de programación Dr Logo. El CPC664 contiene las mismas cosas, pero está incluido todo en la unidad del teclado, que tiene las teclas de diferente color.

CONCLUSION

El 'hardware' del CPC es muy bueno, y muy sólido. Todos los chips usados, excepto el grupo de puertas lógicas, son fáciles de encontrar y son productos probados. El grupo de puertas lógicas, en común con todos los que yo he visto, se calienta bastante. Para ayudarle a disipar algo de calor está sujeto a un pequeño disipador de metal. Debe tener en cuenta este factor de calentamiento cuando use la máquina, especialmente en días calurosos. Intente colocar la máquina en una posición donde no le de la luz directa del sol. No obstruya las ranuras de refrigeración del teclado ni de la parte superior del monitor. El sobrecalentamiento no suele dañar la máquina permanentemente (a menos que realmente sea excesivo) pero cuando la máquina tiene una temperatura excesiva puede fallar repentinamente o hacer cosas inesperadas. Antes de devolver el CPC a su vendedor en un día de verano por errores aleatorios, asegúrese que no está sobrecalentándolo obstruyendo las ranuras de refrigeración.

En el siguiente capítulo veremos el chip central de la máquina, el microprocesador Z80A.

CAPITULO DOS
El Microprocesador Z80

En el corazón de su Amstrad reside el microprocesador Z80. El Z80 está diseñado y fabricado por Zilog Inc. Lleva en el mercado unos ocho años. El diseño del Z80 es en realidad una expansión de su antecesor, el 8080, un producto de Intel Corp., que posteriormente diseñó y comercializó el popular microprocesador 8086. Usando la arquitectura básica del 8080, pero añadiéndole más registros internos y un juego de instrucciones muy expandido, Zilog consiguió producir un microprocesador potente y relativamente barato. (En el momento de escribir este libro se puede adquirir un Z80 por unas 700₧). La versión del Z80 que se usa en las máquinas CPC es el Z80A. (La única diferencia entre un Z80 y un Z80A es la velocidad a la que puede trabajar el chip, el Z80A puede trabajar desde 4MHz de frecuencia de reloj, mientras que el Z80 solamente puede funcionar a 2.5 MHz. Esto significa que el Z80A funciona casi al doble de velocidad).

En este capítulo no intento proporcionarle un conocimiento exhaustivo del microprocesador Z80. Hay muchos libros que se centran exclusivamente en el microprocesador Z80, su juego de instrucciones, y sus aplicaciones. En el apéndice 2 se listan algunas de estas obras. La intención del autor con esta introducción es darle aquella información sobre el Z80 que le permita comprender el 'software' y el 'hardware' que se presenta en los siguientes capítulos. Teniendo en cuenta esto, la presentación del juego de instrucciones puede parecer superflua, pero se ha incluido como una referencia rápida para los lectores que quieran estudiar o modificar los programas en lenguaje ensamblador de los próximos capítulos.

Z80A CONEXIONES Y DESCRIPCIÓN DE SEÑALES

El microprocesador Z80 viene encapsulado en un paquete DIL (Dual In Line) de 40 patillas. Como la mayoría de los otros microprocesadores, los chips Z80 producen todas las señales requeridas para direccionar, leer o escribir en los chips de memoria externos. (Adicionalmente, el Z80 produce señales específicas para un tipo de memoria RAM denominada RAM Dinámica). Este tipo de memoria RAM ofrece una gran capacidad de memoria por chip, pero necesita continuos comandos de refresco, de otra forma perdería su contenido. El Z80 tiene la capacidad de suministrar estas señales de refresco, evitando la necesidad de dispositivos adicionales que realicen esta función, aunque los chips de RAM dinámica del CPC no usan el Z80 para el refresco. En su lugar lo realiza el chip del grupo de puertas lógicas, que se ha explicado en el capítulo uno. El diagrama de conexiones del Z80A se muestra en la figura 2.1. Vamos a ver ahora brevemente el propósito de cada una de las señales, o grupo de señales, que se ven en el diagrama.

Patillas 1 a 5 y 30 a 40: A0-A15: Estas líneas contienen las direcciones de la memoria externa, o de la posición de E/S (Entrada/Salida) a la que está accediendo el microprocesador. Con estas 16 líneas de direccionamiento se puede direccionar hasta 65536 (decimal).

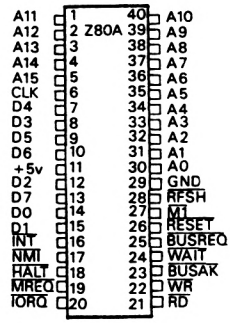


Fig 2.1: Conexiones del Z80A

Patilla 6: Clock. Esta es la patilla a la que se aplica la señal de reloj del microprocesador. La frecuencia de este reloj determina la velocidad a la que se ejecutan las instrucciones. Para el Z80A la máxima frecuencia es 4 MHz.

Patillas 7-10 y 12-15: D0-D7. Estas son las líneas de datos sobre las que se trasfieren los datos desde, y hacia, el Z80. Estas líneas tienen que ser bidireccionales, para permitir la circulación de datos hacia dentro y hacia fuera del microprocesador.

Patilla 11: VCC. Aquí se conecta el polo positivo de la fuente de alimentación. El Z80 necesita +5 voltios a un máximo de 200 miliamperios.

Patilla 16: INT (negativa). Esta es la entrada de la señal de interrupción primaria, la dispara el borde negativo. Cuando aparece un borde negativo en esta entrada, y están permitidas las interrupciones por programa, ejecutando una instrucción EI (Activar Interrupciones), comienza una secuencia de interrupción desde que se completa la instrucción que se está ejecutando. Ninguno de los proyectos que incluye este libro hace uso de las interrupciones. El sistema operativo del CPC usa ampliamente las interrupciones para la temporización de sus rutinas. El Z80 proporciona extensas y complejas opciones para el manejo de las interrupciones. No vamos a entrar en grandes detalles sobre el uso de las interrupciones, ya que no son aplicables los proyectos que tenemos entre manos; los lectores que necesiten conocer algo más acerca del manejo de interrupciones deben referirse a alguno de los libros que hay sobre Z80.
Patilla 17: NMI (negativa). Esta es la señal de interrupción no enmascarable. Como su nombre indica, es una patilla de interrupción cuya función no se puede desactivar por programa (es decir, no se puede enmascarar). Cuando aparece un borde negativo en esta entrada, comienza una secuencia de interrupción desde que se completa la instrucción que se está ejecutando.

Patilla 18: HALT (negativa). Esta es una señal de salida del Z80 que se pone negativa cuando se ejecuta una instrucción HALT.

Patilla 19: MREQ (negativa). Es una señal de salida que se pone negativa cuando el Z80 quiere leer o escribir en una dirección de memo-

ria, es incompatible con la señal IORQ. En la siguiente sección veremos una secuencia de señales para lectura y escritura en memoria.

Patilla 20: IORQ (negativa). Esta salida se pone negativa para indicar que el Z80 quiere leer o escribir en una dirección de E/S. En la siguiente subsección veremos un ciclo de lectura de E/S y de escritura de E/S.

Patilla 21: RD (negativa). Esta es la señal de salida de sincronismo de lectura. Esta línea manda un impulso negativo para sincronizar la lectura desde el Z80 de los datos presentados por la memoria o los dispositivos de E/S.

Patilla 22: WR (negativo). Esta es la señal de salida de sincronización de escritura. Esta línea manda un impulso negativo para sincronizar la escritura de datos desde el Z80 a la memoria o a los dispositivos de E/S.

Patilla 23: BUSAK (negativa). Esta señal de salida se pone negativa para indicar a los dispositivos externos que las líneas de datos(D0-D7) y de dirección(A0-A7) están en estado de alta impedancia. Cuando esto ocurre, otros dispositivos pueden usar estas líneas. Un ejemplo de un dispositivo que puede compartir el 'bus' de esta forma es un controlador de disco con capacidad DMA (Direccionamiento Dinámico de Memoria).

Patilla 24: WAIT (negativa). Esta señal de entrada se usa para extender los ciclos de lectura y escritura indefinidamente, para facilitar el uso de dispositivos o chips de memoria lentos, con el Z80.

Patilla 25: BUSRQ (negativa). Esta señal de entrada la ponen negativa los dispositivos que requieren acceso al 'bus', y que deben esperar hasta que el Z80 complete la instrucción que está ejecutando, respondiendo con un BUSAK cuando esté disponible.

Patilla 26: RESET (negativa). Es una señal de entrada que, cuando se pone negativa, restaura completamente al microprocesador.

Patilla 27: M1 (negativa). La ejecución de cualquier instrucción de un microprocesador requiere lo que se llama 'fetch cycle' o ciclo de obtención. Aquí es donde el microprocesador obtiene la siguiente instrucción a ejecutar. Va seguido de un ciclo de ejecución. Esta señal de salida se pone negativa cuando el Z80 entra en ciclo de obtención. Cuando el Z80 está obteniendo una instrucción de dos octetos, la señal M1 se pondrá negativa hasta que haya leído los dos octetos.

Patilla 28: RFSH (negativa). Esta señal se usa en conjunción con MREQ para refrescar la memoria RAM dinámica. En el CPC no está conectada esta patilla.

Patilla 29: GND. Es la patilla a la que llega la masa o tierra de la fuente de alimentación.

Ahora que hemos visto las señales que entran y salen en el Z80, veremos cómo se usan. En los siguientes ejemplos se ha simplificado el funcionamiento interno del microprocesador.

La secuencia interna del microprocesador Z80 se divide en secciones, que se llaman "ciclos T". en términos generales no nos concierne , ya que solo estamos interesados en cómo se relaciona el Z80 con los

dispositivos externos. Aquellos lectores que quieran buscar más sobre el Z80 deberán escoger entre uno de los muchos buenos libros que sobre Z80 hay en el mercado. Las secuencias de ciclos que presentamos aquí representan una buena base para aprender cómo se comunica el Z80 con los dispositivos externos.

TEMPORIZACIÓN DE SEÑALES

La figura 2.2 nos muestra la temporización de señales de un ciclo de lectura del Z80. Este ocurre cuando el Z80 está leyendo un octeto de datos dentro de un registro interno, desde la memoria. En este caso la secuencia de señales es como sigue:

1)El Z80 coloca en las líneas de direccionamiento A0-A15 la dirección de la posición a la que desea acceder.

2)Después de un pequeño lapso de tiempo, para darle tiempo a que se establezcan las líneas, lanza un impulso negativo de sincronismo por la línea RD. Si la lectura es a una dirección de memoria, pone

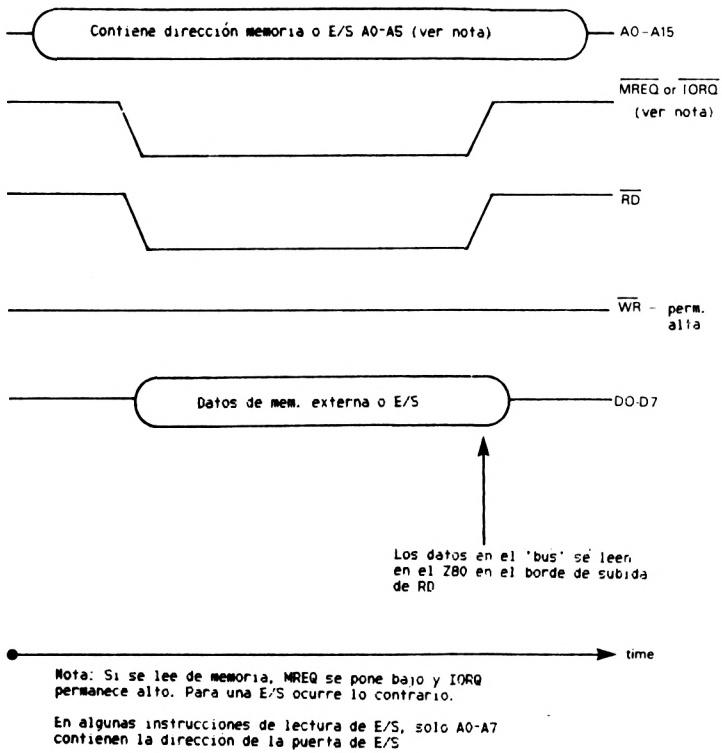


Fig. 2.2: Secuencia de señales de un ciclo de lectura del Z80

negativa la señal de MREQ al mismo tiempo que la RD. Si va a leer de un dispositivo de E/S, pone negativa la señal IORQ al mismo tiempo que la línea RD.

3)Durante el tiempo que está negativa la línea RD, el contenido de la posición de memoria o del registro de E/S que se está direccionando debe poner los datos en las líneas de 'bus' D0-D7. Justo antes de que termine el impulso RD, se leen en el Z80 los datos presentados en el 'bus' de datos.

4)RD y MREQ (o IORQ) vuelven a su valor positivo de inactividad.

5)Las líneas de direccionamiento se liberan. aquí termina el ciclo de lectura.

A los lectores que son nuevos en el tema de los microprocesadores, puede parecerles que esta secuencia es suficientemente compleja como para hacer al Z80 lento. Sin embargo no lo es. En la configuración del Z80A usada en el CPC, la secuencia anterior dura 750 nanosegundos, esto es 0'750 microsegundos, o 0'000750 milisegundos, o 0'000000750 segundos. Para poner una comparación, la media de reacción de un ser humano ante un suceso inesperado tarda cerca de 10 milisegundos, o 0'010 segundos.

El ciclo de escritura en memoria difiere del ciclo de escritura a un dispositivo de E/S, por lo tanto, vamos a considerar los dos en diagramas separados. La figura 2.3 muestra el ciclo de escritura a memoria del Z80. Tiene lugar en los siguientes pasos:

1)El Z80 coloca en las líneas de direccionamiento A0-A15 la dirección de memoria a la que quiere escribir.

2)El Z80 pone negativa la línea MREQ.

3)El Z80 coloca en las líneas del 'bus' de datos D0-D7, los datos que quiere escribir.

4)El Z80 lanza un impulso de sincronismo de escritura por la línea WR.

5)Cuando termina el impulso WR de sincronismo, la línea MREQ vuelve a su estado positivo (inactivo).

6)Finalmente, se restauran las líneas de datos y de direccionamiento.

El registro A o Acumulador: Este registro es en el que se ponen siempre los resultados de las operaciones. Estas operaciones son sumas, restas, y operaciones lógicas como OR, AND, XOR y NOT.

Dispone de seis registros simples, que se pueden usar también como pares, conteniendo cada uno un valor de 16 bits, o una dirección. Estos son los *registros generales* son B, C, D, E, H y L. Cuando se usan como registros de 16 bits, se les llama registros BC, DE y HL y es la instrucción la que determina la forma de uso.

A continuación viene el *registro de indicadores*, o registro F. Aquí es donde almacena el Z80 sus códigos de condición. Estos códigos revelan todo lo que necesita conocer un programa sobre el resultado de una operación aritmética o una comprobación. Por ejemplo: ¿Ha generado un acarreo?, ¿dio cero como resultado?, y así sucesivamente. Esto

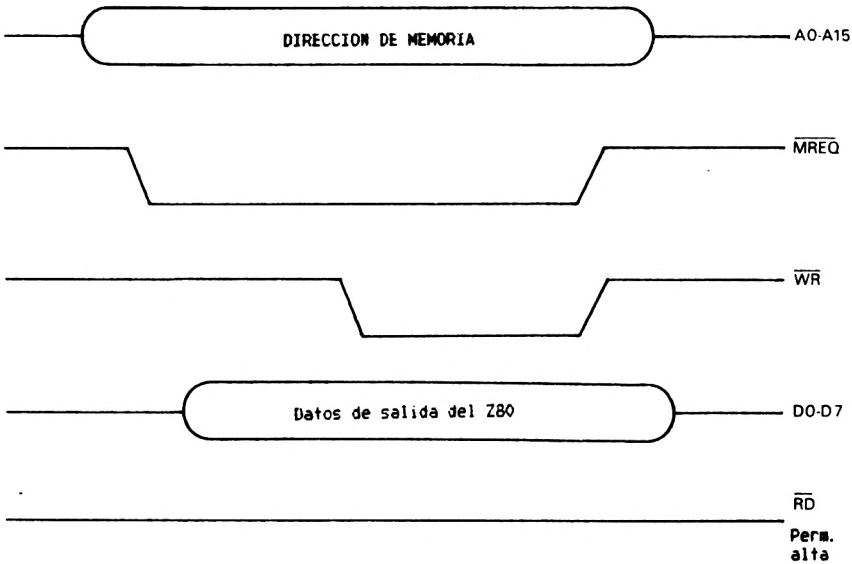
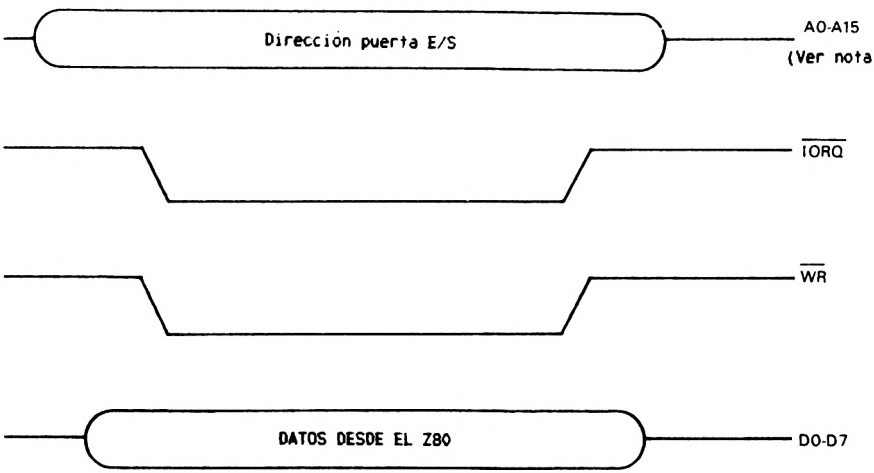


Fig. 2.3: Secuencia de escritura en memoria



Nota: Algunas instrucciones de E/S del Z80 usan solamente A0-A7 para direccionar una puerta de E/S. Usando esta instrucción, los valores de A8-A15 no son útiles.

Fig. 2.4: Secuencia de escritura a un dispositivo de E/S

permite el examen de los códigos de condición antes de ejecutar una bifurcación o un salto condicional. (Si algunos de estos términos resultan extraños para usted, no se apure, pronto encontrará la explicación en la lista de instrucciones).

Todos los microprocesadores tienen un registro que se llama *registro PC*. Significa Program Counter (Contador de Programa). Este registro es de 16 bits y contiene siempre la dirección de la siguiente instrucción a ejecutar.

El *registro SP* es el apuntador de la pila. Para comprender la función y el propósito de un registro SP, debe usted comprender el concepto de pila.

A continuación explicamos la secuencia de un ciclo de escritura a un dispositivo de E/S. Ocurre en los siguientes pasos;(Ver la figura 2.4):

1)El Z80 manda al 'bus' de direcciones la dirección de la puerta de E/S en la que quiere escribir.

2)El Z80 coloca en el 'bus' de datos, DO-D7, los datos que desea escribir.

3)El Z80 pone negativas simultáneamente las líneas IORQ y WR.

4)Las líneas WR y IORQ vuelven a su nivel positivo (inactivo).

5)Se restauran los 'bus' de direcciones y de datos.

Estas son las secuencias de las señales de lectura y escritura de datos en el Z80. Ahora, debemos ver el Z80 desde el punto de vista del programador, ¡por favor guarden los soldadores!

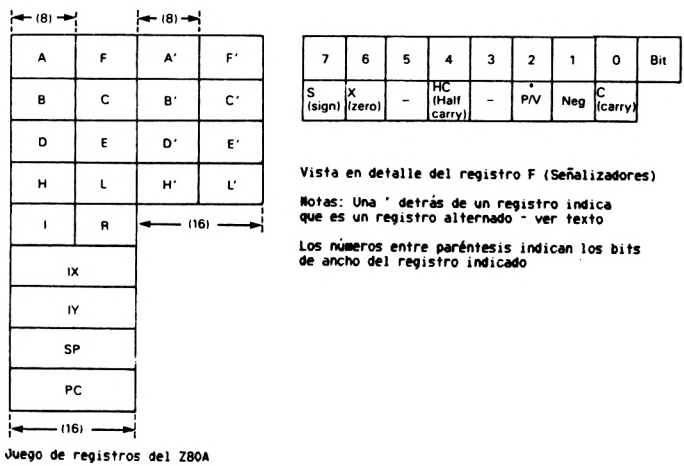


Fig. 2.5:Mapa de los registros del Z80

PROGRAMACIÓN DEL Z80

Desde el punto de vista del programador, el Z80 se ve como en la figura 2.5. Ahora describiremos los registros disponibles para el programador.

> Una *pila* es un tipo de estructura de datos en la que el último en llegar es el primero en salir. Se reserva un rango de posiciones de memoria para su uso en forma de pila. El registro apuntador de pila se carga con la dirección de la posición de memoria más alta de este rango. Cuando el programa necesita salvar datos desde los registros de la CPU durante la ejecución de un programa, usa unas instrucciones que ponen los datos en la pila. Estas instrucciones se llaman PUSH y POP. En una operación PUSH los datos a salvar se escriben dentro de las posiciones de memoria apuntadas por el registro SP, después la CPU decrementa automáticamente el SP (esto es, le resta uno a su valor). La operación POP es la opuesta a PUSH. Cuando se ejecuta una instrucción POP, la CPU incrementa primero el contenido del registro SP (esto es, le suma uno), después carga dentro de los registros que especifica la instrucción, el contenido de la posición de memoria a la que apunta el registro SP. El registro que se va a usar en la instrucción PUSH o POP se especifica en la instrucción. La figura 2.6 nos muestra un programa que hace PUSH del registro A dentro de la pila, después hace POP de él. El programa se lista en lenguaje ensamblador. El lenguaje ensamblador le permite programar directamente el Z80, tomando como referencia el juego de instrucciones. En el capítulo cinco, se detalla e ilustra la programación en lenguaje ensamblador, y en la siguiente sección de este capítulo veremos todo el juego de instrucciones del Z80, del que PUSH y POP son solamente dos.

El ***registro I*** se usa para almacenar el vector de interrupción. Se usa durante una secuencia de interrupción, y contiene los 8 bits superiores de la dirección de los programas que manejan las interrupciones de los dispositivos. Como ya dijimos anteriormente, los proyectos que se presentan en este libro no usan las interrupciones, por lo que los lectores no tendrán ocasión de usar este registro cuando escriban los programas.

Los ***Registros IX e IY*** son dos registros de índice. Son especialmente útiles en operaciones en las que se copia un rango de direcciones de memoria a otro. El contenido de estos registros se usa como dirección, y hay instrucciones para añadir desplazamientos, y también para incrementar y decrementar su contenido. Como veremos más adelante en este libro, esto nos permite programar de forma simple algunas operaciones complejas.

Finalmente el ***registro R***. Este registro contiene la dirección de refresco para la opción de refresco de memoria del Z80. En las máquinas CPC no se usa. El Z80 incrementa constantemente los siete bits inferiores de este registro.

En el interior del Z80 existen unos duplicados de los registros A, B, C, D, E, H, L y F. Estos registros duplicados se conocen como juego de *registros alternados*. Se usa el mismo juego de instrucciones para ambos juegos de registros, pero solo puede usar un juego a la vez. Para comenzar a usar el segundo juego de registros (o volver al juego principal) el programador debe usar la instrucción EXX. Existe también una instrucción para alternar los registros A y F, mientras

```
 100 ; Programa de demostración del apuntador de pila
 200           ENT  $
 300           LD   A,#55        ; Poner un valor conocido en el registro A
 400           PUSH AF           ; PUSH de los registros A y F en la pila.
 500 ; El registro AF está disponible en este punto para que
 600 ; lo use el programador. Cuando haya terminado de usarlo
 700 ; el programa, debe hacer POP de los registros A y F
 800 ; para restaurar su contenido original.
 900           POP AF            ; POP de los registros
1000           RET               ; volver al que lo llamó
```

El programa anterior está escrito en lenguaje ensamblador. DEVPAC es uno de los paquetes ensambladores que hay disponibles para el CPC. Un ensamblador toma programas como el que hemos visto y los convierte en instrucciones de código máquina, como la hace el intérprete BASIC en su CPC. En lenguaje ensamblador cada línea tiene un número - como en BASIC. Cualquier cosa que aparezca detrás de un ";", es un comentario - es análogo a la sentencia REM del BASIC. "ENT $" en la línea 200 es una directriz del ensamblador. Esta en particular le dice al ensamblador que tome la línea 300 como principio - o como punto de ENTrada del programa. Usted no podrá ejecutar este programa, a menos que haya usado previamente el ensamblador para convertirlo en código máquina. (En el capítulo 5 se explican los métodos que se usan para ello).

En la siguiente explicación, "(SP)-1" representa la dirección de memoria que hay por debajo de la apuntada por el registro SP, mientras que "(SP)" es la dirección apuntada por el registro SP. Y "(SP)+1" es la dirección por encima de la apuntada por el registro SP.

Refiérase a la descripción de la operación de la pila que hemos dado en el texto. El contenido del registro A, del registro SP y de la memoria apuntada por el registro SP, será como sigue, DESPUES de la ejecución de cada línea numerada:

Línea 200: A y F=xx (Donde xx significa imprevisible). Asumimos que SP contiene Hex FC00.
(SP), (SP)+1 contienen el valor salvado del PC, del programa que está ejecutando una instrucción CALL para llamar a esta subrutina de demostración de la pila. Esta dirección se llamará RET PC ya que es la dirección que se debe poner en el registro PC cuando se ejecute la siguiente instrucción RET.

Línea 300: A=Hex 55. F=xx. SP=FC00. (SP)+1,(SP)+2=RET PC

Línea 400: A=55. F=xx. SP=FBFE. (SP)=xx (registro F salvado). (SP)+1=55 (registro A salvado).(SP)+2,(SP)+3=RET PC

Línea 900: A=55. F=xx. SP=FC00. (SP)+1=RET PC

Línea 1000: A=55. F=xx. SP=FC02. (SP),(SP)+1=xx

Fig. 2.6 Uso de las instrucciones PUSH y POP y de la pila

que se continúan usando los mismos registros generales. Los registros SP, IY, IX, R, I y PC permanecen ajenos a estos cambios.

Una vez que hemos visto el juego de registros del Z80, vayamos a ver el juego de instrucciones.

El juego de instrucciones de un microprocesador es la lista de instrucciones que puede ejecutar. En este contexto una instrucción significa un código que interpreta el microprocesador como algo así; "Mueve el contenido del registro A al registro H", otro código puede querer decir; "suma el contenido del registro A al contenido del registro L y coloca el resultado dentro del registro A". A estos códigos se les llama códigos de operación, abreviados generalmente como 'opcodes'. Cuando colocamos algunas de estas instrucciones en la memoria y le decimos al microprocesador que comience a ejecutarlo, estamos escribiendo y ejecutando un programa. Todos los programas que funcionan en un microprocesador consisten en una lista de códigos de operación en memoria. Esto sigue siendo válido cuando escribimos un programa en un lenguaje como el Locomotive BASIC de su CPC, o cuando usted ejecuta instrucción por instrucción con un ensamblador, como el DEVPAC que aprenderemos en el capítulo 5. Es obvio que cuantas más instrucciones hay disponibles en un microprocesador más potente resulta este. El Z80 ha sido virtualmente insuperable como el mejor de los microprocesadores de 8 bits en el mundo, principalmente por la potencia de su juego de instrucciones. El Z80 tiene 158 tipos de instrucciones. En las siguiente páginas las veremos brevemente. Ya que esto es solamente una introducción el Z80, no vamos a entrar en grandes detalles sobre ellas, los lectores que quieran ampliar sus conocimiento en esta materia se deberán dirigir a alguno de los muchos libros que hay sobre la programación del Z80.

JUEGO DE INSTRUCCIONES DEL Z80 EN ORDEN ALFABÉTICO

La siguiente es una lista completa del juego de instrucciones del Z80. El formato que se usa en cada instrucción es:

Número) **Nemotécnico de la instrucción:** Descripción

INDICADORES:lista de indicadores afectados

El número que precede a cada instrucción de la lista sirve para poder referirnos a cada una fácilmente en otros capítulos. Se usan muchas abreviaturas y símbolos en la descripción del juego de instrucciones y tienen el siguiente significado:

s=Una posición donde poner o tomar datos. Se listan todas las posibilidades en la descripción de la instrucción.

r=cualquier registro A,B,C,D,E,H,L

rp=cualquier par de registros BC,DE,HL

b=un número de bit, de 0 a 7 solamente

xx=cualquier valor hexadecimal

(r)=la dirección de memoria apuntada por el registro r. Ej.(B) significa la dirección apuntada por el registro B, mientras que (HL) es la dirección apuntada por el par de registros HL.

(IY+xx)=La dirección a la que apunta el resultado del cálculo IY+xx. xx será normalmente el tercer octeto situado detrás del código de operación.

INDICADORES afectados,- es la lista de los indicadores que pueden ser alterados por la instrucción. los indicadores no mencionados no resultarán alterados nunca por la instrucción.

Muchas instrucciones realizan acciones si se cumplen ciertas condiciones, en la sección de descripciones, la sección "cc" debe ser reemplazada por uno de los nemotécnicos de condición que tienen el siguiente significado:

Condición	*significado*
Z	Indicador de cero a uno.
NZ	Indicador de cero a cero.
C	Indicador de acarreo a uno.
NC	Indicador de acarreo a cero.
PO	Indicador P/V a cero.
PE	Indicador P/V a uno.
P	Indicador de signo (S) a uno.
M	Indicador de signo a cero.

Para poner un ejemplo, JP cc,xxxx (Salto condicional), instrucción 61 de la lista, puede modificar su cc a una de las siguientes condiciones:

```
JP  Z,xxxx    =Salta a xxxx si el indicador de cero está a 1
JP  NZ,xxxx   =Salta a xxxx si el indicador de cero está a 0
JP  C,xxxx    =Salta a xxxx si el indicador de acarreo está a 1
JP  NC,xxxx   =Salta a xxxx si el indicador de acarreo está a 0
JP  PO,xxxx   =Salta a xxxx si el indicador P/V está a 0
JP  PE,xxxx   =Salta a xxxx si el indicador P/V está a 1
JP  P,xxxx    =Salta a xxxx si el indicador S está a 1
JP  M,xxxx    =Salta a xxxx si el indicador S está a 0
```

Lista de instrucciones del Z80

1) **ADC A,s** El operando s y el bit de acarreo se suman al contenido del registro A. El operando s puede ser cualquiera de los siguientes; xx, r, (HL), (IX+xx) o (IY+xx). El resultado se almacena en A.
   INDICADORES:TODOS

2) **ADC HL,rp** Los dos pares de registros HL y rp se suman junto con el bit de acarreo. El resultado se almacena en HL. rp puede ser BC,DE, HL o SP.
   INDICADORES:S,Z,H,P/V,C

3) ADD A,(HL) El contenido de la posición de memoria apuntada por el registro HL se suma al contenido del registro A. El resultado se almacena en el registro A.
INDICADORES:S,Z,H,P/V,C

4) ADD A,(IX+xx) El contenido de la posición de memoria cuya dirección se obtiene sumando xx al contenido del registro IX, se suma al contenido del registro A. El resultado se almacena en A. IX permanece sin alterar.
INDICADORES:S,Z,H,P/V,C

5) ADD A,(IY+xx) El contenido de la dirección de memoria cuya dirección se obtiene sumando xx al contenido del registro IY, se suma al contenido del registro A. El resultado se almacena en A. IY permanece sin alterar.
INDICADORES:S,Z,H,P/V,C

6) ADD A,n Suma el octeto de datos n al registro A. El resultado se almacena en A.
INDICADORES:S,Z,H,P/V,C

7) ADD A,r Suma el contenido del registro r al registro A. El resultado se almacena en A.
INDICADORES:S,Z,H,P/V,C

8) ADD HL,rp Suma el contenido del par de registros rp al par de registros HL, y coloca el resultado en el par de registros HL. rp puede ser BC, DE, HL o SP.
INDICADORES:H,N,C

9) ADD IX,rp Suma el contenido del registro IX al par de registros rp. El resultado se sitúa en el registro IX. rp puede ser BC, DE, IX o SP.
INDICADORES:H,N,C

10) ADD IY,rp Suma el contenido del registro IY al par de registros rp. El resultado se sitúa en el registro IY. rp puede ser BC, DE, IY o SP.
INDICADORES:H,N,C

11) AND s Hace un AND lógico entre el registro A y el operando s. El resultado se coloca en el registro A. s puede ser r, xx, (HL),(IX+xx) o (IY+xx).
INDICADORES:TODOS

12) BIT b,(HL) Comprueba el bit especificado por b en la posición de memoria apuntada por el par de registros HL. El indicador Z se limpia o se pone de acuerdo con el resultado. Afecta también a la mayoría de los otros indicadores.
INDICADORES:S,Z,H,P/V,N

13) BIT B,(IX+xx) Comprueba el bit especificado de la posición de memoria apuntada por el cálculo (IX+xx), y el indicador Z se limpia o se pone para indicar su estado.
INDICADORES:S,Z,H,P/V,N

14) BIT B,(IY+xx) Comprueba el bit especificado de la posición de memoria apuntada por el cálculo (IY+xx), y el indicador Z se limpia o se pone para indicar su estado.
INDICADORES:S,Z,H,P/V,N

15) BIT b,r Comprueba el bit especificado del registro r, y se pone el indicador Z de acuerdo con el estado del bit.
INDICADORES:S,Z,H,P/V,N

16) CALL cc,xx Se comprueba la condición especificada, y si es cierta, se almacena el contenido de PC en la pila, se coloca el valor de xxxx en PC para ejecutar la subrutina. Se usa RET para retornar de la subrutina. Vea la introducción de esta sección para obtener la lista de condiciones que se pueden comprobar en la parte cc de esta instrucción.
INDICADORES:NINGUNO

17) CALL xxxx Es similar a la instrucción anterior, pero siempre efectúa la llamada, no es condicional.
INDICADORES:NINGUNO

18) CCF Complementa (invierte) el estado del indicador de acarreo
INDICADORES:H,N,C

19) CP s Compara el contenido del registro A con el operandos. El operando s puede ser r, xx, (HL), (IX+xx), (IY+xx) El resultado de la comparación se usa solamente para poner los indicadores, el resultado real no se almacena en ninguna parte.
INDICADORES:TODOS

20) CPD Se resta el contenido de la posición de memoria apuntada por el registro HL del contenido del registro A. El resultado no se almacena. Después se decrementan los registros HL y BC.
INDICADORES:S,Z,H,P/V,N,C

21) CPDR Esta instrucción es casi igual que CPD pero con una variante importante, la instrucción se repite automáticamente hasta que BC llega a cero, o el contenido del registro A coincide con el de la dirección a la que apunta el registro HL. Una cosa a tener en cuenta es que el registro BC se decrementa antes de ejecutarse la instrucción.
INDICADORES:S,Z,H,P/V,N

22) CPI Se resta de A el contenido de la posición de memoria a la que apunta el par de registros HL. El resultado no se almacena. Después se incrementa el registro HL, y se decrementa el registro BC.
INDICADORES:S,Z,H,P/V,N

23) CPIR Esta es una versión automática de CPI. Su funcionamiento es similar a excepción de que se repite automáticamente hasta que BC llega a cero o el contenido de la posición de memoria a la que apunta HL coincide

con el contenido del registro A. BC se incrementa antes de la ejecución.
INDICADORES:S,Z,H,P/V,N

24) CPL Complementa (invierte) el contenido del registro A.
INDICADORES:H,N

25) DAA Esta instrucción cambia el contenido del registro A a su equivalente BCD (Decimal Codificado en Binario). Esto es, hace un ajuste decimal del contenido del registro A.
INDICADORES:S,Z,H,P/V,C

26) DEC s Decrementa s, esto es, resta uno a su valor. s puede ser: r, (HL), (IX+xx) o (IY+xx).
INDICADORES:S,Z,H,P/V,N

27) DEC rp Se decrementa el par de registros seleccionado. s puede ser: BC,DE,HL o SP.
INDICADORES:NINGUNO

28) DEC IX Se decrementa el registro IX.
INDICADORES:NINGUNO

29) DEC IY Se decrementa el registro IY.
INDICADORES:NINGUNO

30) DI Desactiva las interrupciones. Después de la ejecución de esta instrucción, se ignoran todas las peticiones de interrupción que llegan a la patilla INT del Z80. Las peticiones de interrupción que lleguen a la patilla NMI si se ejecutan.
INDICADORES:NONE

31) DJNZ xx Se decrementa el registro B. Si no contiene cero, se suma el valor de xx al valor de PC, ejecutando un salto. La suma se hace usando la aritmética de complemento a dos, esto significa que el bit más alto del octeto xx se toma como positivo si es cero, y como negativo si es uno. Los otros 7 bits se suman o restan del valor de PC.
INDICADORES:NINGUNO

32) EI Activa la respuesta del Z80 a las peticiones de interrupción de la patilla INT. Esto se realiza después de ejecutar la instrucción.
INDICADORES:NINGUNO

33) EX AF,AF' Se intercambia el contenido de AF con el del juego alternado A' y F'. Realmente lo que hace es seleccionar los registros A' y F' para su uso.
INDICADORES:TODOS

34) EX DE,HL Se intercambian los contenidos de los registros DE y HL.
INDICADORES:NINGUNO

35) EX (SP),HL Se intercambian el contenido del registro L con el de la posición de memoria apuntada por SP, y el contenido del registro H con el de la posición de memoria apuntada por (SP+1).
INDICADORES:NINGUNO

36) EX (SP),IX Los 8 bits inferiores del registro IX se intercambian con el contenido de la posición de memoria apuntada por SP, mientras que los 8 bits superiores lo hacen con el contenido de la posición de memoria apuntada por (SP+1).
INDICADORES:NINGUNO

37) EX (SP),IY Los 8 bits inferiores del registro IY se intercambian con el contenido de la posición de memoria apuntada por SP, mientras que los 8 bits superiores lo hacen con el contenido de la posición de memoria apuntada por (SP+1).
INDICADORES:NINGUNO

38) EXX Se intercambia el contenido de todos los registros generales con el de los alternados. Los registros afectados son BC, DE y HL.
INDICADORES:NINGUNO

39) HALT Se para la CPU. Se queda ejecutando NOPs hasta que ocurre una interrupción o se la restaura externamente
INDICADORES:NINGUNO

40) IM 0 Pone el modo de interrupción 0.
INDICADORES:NINGUNO

41) IM 1 Pone el modo de interrupción 1.
INDICADORES:NINGUNO

42) IM 2 Pone el modo de interrupción 2.
INDICADORES:NINGUNO

43) IN r,(C) Lee en el registro A, el contenido de la puerta de E/S cuya dirección de 16 bits está en el registro BC. El registro C proporciona los bits de dirección A0-A7 y el registro B proporciona los bits A8-A15.
INDICADORES:S,Z,H,P/V,N

44) IN A,(xx) Carga en el registro A el contenido del registro de E/S apuntado por el octeto xx. Como el campo xx es un solo octeto, y se usa para suministrar los bits A0-A7 los bits A8-A15 son suministrados por el contenido original del registro A. En el CPC es mejor usar la instrucción IN r,(C).
INDICADORES:NINGUNO

45) INC r Incrementa el contenido del registro r.
INDICADORES:S,Z,H,P/V,N

46) INC rp Incrementa el par de registros seleccionado. El rp puede ser BC, DE, HL o SP.
INDICADORES:NINGUNO

47) INC (HL) Se incrementa el contenido de la posición de memoria apuntada por el registro HL, y se almacena en el mismo lugar.
INDICADORES:S,Z,H,P/V,N

48) INC (IX+xx) Se incrementa el contenido de la posición de memoria apuntada por (IX+xx), y se almacena en el mismo lugar.
INDICADORES:S,Z,H,P/V,N

49) INC (IY+xx) Se incrementa el contenido de la posición de memoria apuntada por (IY+xx), y se almacena en el mismo lugar.
INDICADORES:S,Z,H,P/V,N

50) INC IX Incrementa el contenido del registro IX.
INDICADORES:NINGUNO

51) INC IY Incrementa el contenido del registro IY.
INDICADORES:NINGUNO

52) IND Se escribe la dirección de E/S a la que apunta el registro C sobre la posición de memoria apuntada por el registro HL.Después se decrementan los registros B y HL.
INDICADORES:S,Z,H,P/V,N

53) INDR Es similar a la anterior, excepto que la instrucción se epite hasta que el registro B llega a cero.
INDICADORES:S,Z,H,P/V,N

54) INI Se escribe el contenido del registro de E/S al que apunta el registro C, sobre la dirección de memoria a la que apunta el registro HL. Después se decrementa el registro B, y se incrementa el registro HL. Durante el direccionamiento del registro de E/S, el registro C contiene los bits A0-A7 de la dirección, y el registro B, que se usa como contador, proporciona los bits A8-A15. Esto limita la utilidad de esta instrucción en el CPC.
INDICADORES:S,Z,H,P/V,N

55) INIR Es la versión automática de INI. El contenido del registro de E/S al que apunta el registro C se transfiere a la posición de memoria apuntada por el registro HL. Después se incrementa el registro HL, y se decrementa el registro B. Si el registro B no ha llegado a cero, se repite la instrucción automáticamente
INDICADORES:S,Z,H,P/V,N

56) JP cc,xxxx Si es verdadera la condición especificada, salta a la dirección xxxx. El campo cc puede ser una de las condiciones listadas al principio de esta sección.
INDICADORES:NINGUNO

57) JP xxxx Salta a la posición xxxx. Este salto no es condicional, por lo que ocurrirá siempre.
INDICADORES:NINGUNO

58) JP (HL) Salta a la posición de memoria apuntada por el registro HL.
INDICADORES: NINGUNO

59) JP (IX) Salta a la posición de memoria apuntada por el registro IX.
INDICADORES: NINGUNO

60) JP (IY) Salta a la posición de memoria apuntada por el registro IY.
INDICADORES: NINGUNO

61) JR cc,xx Si es verdadero el código de condición, salta a la dirección que se obtiene de sumar xx en complemento a dos al PC. (Ver instrucción 31).
INDICADORES: NINGUNO

62) JR xx Salta a la dirección que se obtiene de sumar xx en complemento a dos al PC. (Ver instrucción 31).
INDICADORES: NINGUNO

63) LD rp,(xxxx) Carga en el registro rp el contenido de la memoria apuntada por xxxx. El registro rp puede ser BC, DE, HL o SP.
INDICADORES: NINGUNO

64) LD rp,xxxx Carga en el registro rp el valor xxxx. El registro rp puede ser BC, DE, HL o SP.
INDICADORES: NINGUNO

65) LD r,xx El registro r se carga con el valor xx. r puede ser A, B, C, D, E, H o L.
INDICADORES: NINGUNO

66) LD r,r' Se carga el contenido del registro r' en el registro r. r y r' pueden ser A, B, C, D, E, H o L.
INDICADORES: NINGUNO

67) LD (BC),A Se escribe el contenido del registro A en la posición de memoria a la que apunta el registro BC.
INDICADORES: NINGUNO

68) LD (DE),A Se escribe el contenido del registro A en la posición de memoria a la que apunta el registro DE.
INDICADORES: NINGUNO

69) LD (HL),xx Se escribe el valor xx en la posición de memoria a la que apunta el registro HL.
INDICADORES: NINGUNO

70) LD (HL),r Se escribe el contenido del registro r en la posición de memoria a la que apunta el registro HL.
INDICADORES: NINGUNO

71) LD r,(IX+xx) Se carga en el registro r el contenido de la posición de memoria apuntada por (IX+xx).
INDICADORES: NINGUNO

72) LD r,(IY+xx) Se carga en el registro r el contenido de la posición de memoria apuntada por (IY+xx).
INDICADORES:NINGUNO

73) LD (IX+xx),xx Se carga la posición de memoria apuntada por (IX+xx) con el valor de xx.
INDICADORES:NINGUNO

74) LD (IY+xx),xx Se carga la posición de memoria apuntada por (IY+xx) on el valor de xx.
INDICADORES:NINGUNO

75) LD (IX+xx),r Se carga la posición de memoria apuntada por (IX+xx) con el contenido del registro r.
INDICADORES:NINGUNO

76) LD (IY+xx),r Se carga la posición de memoria apuntada por (IY+xx) con el contenido del registro r.
INDICADORES:NINGUNO

77) LD A,(xxxx) Se carga el registro A con el contenido de la posición de memoria xxxx.
INDICADORES:NINGUNO

78) LD (xxxx),A Se carga la posición de memoria xxxx con el contenido del registro A.
INDICADORES:NINGUNO

79) LD (xxxx),rp Se carga la posición de memoria xxxx con el contenido del registro inferior del par rp. Y la posición siguiente con el contenido del registro alto de rp.
INDICADORES:NINGUNO

80) LD (xxxx),HL Se carga la posición de memoria xxxx con el contenido del registro L, y la posición siguiente con el contenido del registro H.
INDICADORES:NINGUNO

81) LD (xxxx),IX Se carga la posición de memoria xxxx con el contenido del octeto inferior del registro IX, y la posición siguiente con el contenido del octeto superior.
INDICADORES:NINGUNO

82) LD (xxxx),IY Se carga la posición de memoria xxxx con el contenido del octeto inferior del registro IY, y la posición siguiente con el contenido del octeto superior.
INDICADORES:NINGUNO

83) LD A,(BC) Carga el registro A con el contenido de la posición de memoria apuntada por el registro BC.
INDICADORES:NINGUNO

84) LD A,(DE) Carga el registro A con el contenido de la posición de memoria apuntada por el registro DE.
INDICADORES:NINGUNO

85) LD A,I Carga el registro A con el contenido del registro del vector de interrupción I.
INDICADORES:S,Z,H,P/V,N

86) LD I,A Carga el registro del vector de interrupción con el contenido del registro A.
INDICADORES:NINGUNO

87) LD A,R Carga el registro A con el contenido del registro R.
INDICADORES:S,Z,H,P/V,N

88) LD HL,(xxxx) Carga en el registro L el contenido de la posición de memoria apuntada por xxxx, y en el registro H el contenido de la dirección apuntada por xxxx+1
INDICADORES:NINGUNO

89) LD IX,xxxx Carga el valor xxxx en el registro IX.
INDICADORES:NINGUNO

90) LD IX,(xxxx) Carga en el octeto inferior del registro IX con el contenido de la posición de memoria apuntada por xxxx, y la parte alta con el contenido de la dirección apuntada por xxxx+1
INDICADORES:NINGUNO

91) LD IY,xxxx Carga el valor xxxx en el registro IY.
INDICADORES:NINGUNO

92) LD IY,(xxxx) Carga en el octeto inferior del registro IY con el contenido de la posición de memoria apuntada por xxxx, y la parte alta con el contenido de la dirección apuntada por xxxx+1
INDICADORES:NINGUNO

93) LD R,A Carga en el registro R (refresco) el contenido del registro A.
INDICADORES:NINGUNO

94) LD SP,HL Carga el registro apuntador de pila SP con el contenido del registro HL.
INDICADORES:NINGUNO

95) LD SP,IX Carga el registro apuntador de pila SP con el con nido del registro IX.
INDICADORES:NINGUNO

96) LD SP,IY Carga el registro apuntador de pila SP con el contenido del registro IY.
INDICADORES:NINGUNO

97) LDD Carga el contenido de la posición de memoria apuntada por el registro HL en la posición de memoria apuntada por el registro DE, y después decrementa los registros BC, DE y HL.
INDICADORES:H,P/V,N

98) LDDR Otra instrucción automática. Esta hace lo mismo que LDD pero se repite hasta que BC llega a cero.
INDICADORES:H,P/V,N

99) LDI Carga el contenido de la posición de memoria apuntada por el registro HL en la posición de memoria apuntada

por el registro DE, y después incrementa los registros DE y HL, mientras que decrementa el registro BC.
INDICADORES:H,P/V,N

100) LDIR Esta es la versión automática de LDI. La instrucción tiene el mismo efecto, pero se ejecuta repetidamente hasta que el registro BC llega a cero.
INDICADORES:H,P/V,N

101) LD r,(HL) Se carga en el registro r el contenido de la posición de memoria apuntada por HL.
INDICADORES:NINGUNO

102) NEG Esta instrucción niega el contenido del registro A. Es decir cambia los bits que están a uno por cero y viceversa.
INDICADORES:TODOS

103) NOP No operación, esta instrucción se usa para que el Z80 no haga nada, o para perder tiempo en bucles de aplicaciones con temporizaciones críticas.
INDICADORES:NINGUNO

104) OR s Hace un OR lógico entre el registro A y s. s puede ser: r, xx, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

105) OTDR Se decrementan los registros B y HL. Después se manda el contenido de la posición de memoria apuntada por HL a la puerta de E/S direccionada por el contenido del registro C.
INDICADORES:S,Z,P/V,H,N

106) OTIR Se decrementa el registro HL, y se incrementa el registro B. Después se manda el contenido de la posición de memoria apuntada por HL a la puerta de E/S direccionada por el contenido del registro C.
INDICADORES:S,Z,P/V,H,N

107) OUT (C),r Se manda el contenido del registro r a la puerta de E/S apuntada por el registro BC.
INDICADORES:NINGUNO

108) OUT (xx),A Se manda el contenido del registro A a la puerta de E/S cuya dirección de 8 bits está apuntada por xx.
INDICADORES:NINGUNO

109) OUTD Se manda a la puerta de E/S apuntada por el contenido del registro C, el contenido de la posición de memoria apuntada por el registro HL. Después de esto, se decrementan los registros B y HL.
INDICADORES:S,Z,P/V,H,N

110) OUTI Se manda a la puerta de E/S apuntada por el contenido del registro C, el contenido de la posición de memoria apuntada por el registro HL. Después de esto, se decrementa el registro B y se incrementa el registro HL.
INDICADORES:S,Z,H,P/V,N

111) **POP rp** El contenido de la posición de memoria apuntada por SP se carga en el octeto inferior del registro rp, después se incrementa SP y se carga en el octeto superior de rp el contenido de la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

112) **POP IX** El contenido de la posición de memoria apuntada por SP se carga en el octeto inferior del registro IX, después se incrementa SP y se carga en el octeto superior de IX el contenido de la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

113) **POP IY** El contenido de la posición de memoria apuntada por SP se carga en el octeto inferior del registro IY, después se incrementa SP y se carga en el octeto superior de IY el contenido de la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

114) **PUSH rp** Se decrementa el registro SP, y se escribe el contenido del registro alto del par rp en la posición de memoria apuntada por SP. Después se decrementa SP de nuevo y se almacena el registro inferior del par rp en la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

115) **PUSH IX** Se decrementa el registro SP, y se escribe el contenido de octeto alto de IX en la posición de memoria apuntada por SP. Después se decrementa SP de nuevo y se almacena el octeto inferior de IX en la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

116) **PUSH IY** Se decrementa el registro SP, y se escribeel contenido de octeto alto de IY en la posición de memoria apuntada por SP. Después se decrementa SP de nuevo y se almacena el octeto inferior de IY en la posición de memoria a la que apunta SP.
INDICADORES:NINGUNO

117) **RES b,s** Pone a cero (restaura) el bit b de la posición s. El valor de b debe ser un número entre 0 y 7, mientras que s debe ser uno de los siguientes: r, (HL), (IX+xx) o (IY+xx).
INDICADORES:NINGUNO

118) **RET** Retorno desde una subrutina. El contador de programa se carga desde la pila (ver la explicación de POP), y con ello efectúa un retorno al programa que introdujo el antiguo valor de PC en la pila.
INDICADORES:NINGUNO

119) **RET cc** Si se cumple el código de condición, retorna de la ejecución de la subrutina. (Ver el principio de esta sección para las condiciones cc).
INDICADORES:NINGUNO

120) **RETI** Retorno de una interrupción. El PC de retorno se toma de la pila como se describió anteriormente. Como esta instrucción debe ir siempre al final de una rutina de servicio de interrupción, casi siempre va precedida por la instrucción EI (activar interrupciones). Esto permite que las futuras interrupciones puedan interrumpir al programa principal.
INDICADORES:NINGUNO

121) **RETN** Retorno de una interrupción NMI.
INDICADORES:NINGUNO

122) **RL s** Gira hacia la izquierda a través del bit de acarreo del registro F. El contenido de s se gira un bit hacia la izquierda, y el contenido del bit 7 se desplaza dentro del bit de acarreo en el registro F, y el estado previo del bit de acarreo se desplaza al bit 0 de s. El operando s puede ser: r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

123) **RLA** Gira el registro A hacia la izquierda a través del bit de acarreo. Esta función es idéntica a RL s, excepto que es siempre con el registro A.
INDICADORES:H,N,C

124) **RLCA** El registro A se gira hacia la izquierda y el bit de acarreo del registro F queda con el estado del bit 7 de A antes del desplazamiento.
INDICADORES:H,N,C

125) **RLC r** Como RLCA, excepto que el registro que gira puede ser: A, B, C, D, E, H o L.
INDICADORES:TODOS

126) **RLC (HL)** Se realiza un giro hacia la izquierda, de la forma descrita para RLCA, en la posición de memoria apuntada por el registro HL.
INDICADORES:TODOS

127) **RLC (IX+xx)** Se realiza un giro hacia la izquierda, de la forma descrita para RLCA, en la posición de memoria apuntada por (IX+xx).
INDICADORES:TODOS

128) **RLC (IY+xx)** Se realiza un giro hacia la izquierda, de la forma descrita para RLCA, en la posición de memoria apuntada por (IY+xx).
INDICADORES:TODOS

129) **RLD** Giro decimal hacia la izquierda. Esta instrucción mueve el medio octeto inferior del contenido de la dirección apuntada por HL, a la posición del medio octeto superior, el valor original del medio octeto superior de la posición de memoria se mueve al lugar del medio octeto inferior del registro A, y éste se mueve al medio octeto inferior de la posición de memoria.
INDICADORES:S,Z,H,P/V,N

130) **RR s** La posición s se gira hacia la derecha a través del bit de acarreo del registro F. El giro se realiza de la siguiente forma: La posición seleccionada se desplaza un bit hacia la derecha poniendo el bit 0 en el de acarreo.del registro F, mientras que el bit 7 de la posición seleccionada toma el valor anterior del bit de acarreo. s puede ser: r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

131) **RRA** Giro del registro A a través del bit de acarreo. El giro se realiza como en RR s, con la diferencia de que en esta instrucción A toma el lugar de s como posición girada.
INDICADORES:H,N,C

132) **RRC s** Giro hacia la derecha. El contenido de la posición seleccionada se gira hacia la izquierda de la siguiente manera: Se gira hacia la derecha el contenido del octeto seleccionado. El bit siete toma el valor que tenía el bit 0. El bit de acarreo siempre toma el valor que tenía el bit cero antes del desplazamiento. s puede ser: r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

133) **RRCA** Gira el registro A hacia la derecha. Es como la anterior RRC, excepto que el registro A toma el lugar de s.
INDICADORES:H,N,C

134) **RRD** Giro decimal hacia la derecha. Esta instrucción mueve el medio octeto inferior del registro A al medio octeto superior de la dirección apuntada por HL, en la misma posición de memoria el medio octeto superior reemplaza al medio octeto inferior, mientras que el valor original del medio octeto inferior reemplaza el medio octeto inferior del registro A.
INDICADORES:S,Z,H,P/V,N

135) **RST p** Rearrancar en p. Esta instrucción hace que el contenido de PC se almacene en la pila, y el valor especificado por p se coloca en PC. La siguiente instrucción se ejecuta en la dirección p que puede ser una los siguientes valores: 00,08,10,18,20,28,30 o 38.
INDICADORES:NINGUNO

136) **SBC A,s** Se restan de A el bit de acarreo más el contenido de s, y se coloca el resultado en el registro A. s puede ser: r, xx, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

137) **SBC HL,rp** Se le suma el bit de acarreo al contenido del par de registros rp y el total se resta del contenido del registro HL. El resultado se almacena de nuevo en el registro HL.
INDICADORES:TODOS

138) **SCF** Pone el indicador de acarreo.
INDICADORES:,H,N,C

139) **SET b,s** Pone a uno el bit b de la posición s. El valor de b puede ser entre 0 y 7. La posición seleccionada por s puede ser: (HL), (IX+xx) o (IY+xx).
INDICADORES:NINGUNO

140) **SLA s** Desplazamiento aritmético hacia la izquierda en la posición seleccionada por s. Se desplaza el contenido de la posición seleccionada hacia la izquierda, y se pone el bit 0 a cero, mientras que el bit 7 se desplaza al bit de acarreo del registro F. s puede ser r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

141) **SRA s** Desplazamiento aritmético hacia la derecha en la posición seleccionada por s. Se desplaza el contenido de la posición seleccionada hacia la derecha, el valor original del bit cero se pone en el bit de acarreo. s puede ser r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

142) **SRL s** Desplazamiento lógico hacia la derecha de la posición seleccionada por s. El contenido de s se desplaza un bit hacia la derecha. el bit 7 se pone a cero, y el estado del bit 0 se desplaza dentro del bit de acarreo en el registro F. s puede ser una de las posiciones listadas en SLA.
INDICADORES:TODOS

143) **SUB s** Resta s del registro A. El operando se resta del registro A, y el resultado se almacena de nuevo en el registro A. s puede ser uno de; r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

144) **XOR s** OR exclusivo del registro A con la posición seleccionada por s. El resultado se vuelve a almacenar en el registro A. s puede ser uno de; r, (HL), (IX+xx) o (IY+xx).
INDICADORES:TODOS

CONCLUSIÓN

Como puede ver, el juego de instrucciones del Z80 es realmente largo, especialmente cuando toma en cuenta las variantes de cada instrucción estándar. Use esta sección como referencia, cuando comience a examinar algunos de los programas que se presentan en el libro para manejar los proyectos 'hardware'. Un punto final sobre el Z80: hay algunas de estas instrucciones que no son muy usadas en el CPC. Esto se debe a que no hacen lo que se espera de ellas, debido a que interfieren en el sistema operativo que hay dentro del CPC, haciéndolo fallar. Usted no puede dañar su ordenador si las usa, pero hará cosas extrañas, y tendrá que apagarlo y encenderlo de nuevo para restaurarlo. Estas instrucciones se tratarán en el capítulo 5, que explica la programación en ensamblador, usando el ensamblado DEVPAC. Con esto concluimos la descripción del Z80A.

CAPITULO TRES
Proyectos 'Hardware'

En esta capítulo veremos algunos proyectos 'hardware' que usted puede construir para mejorar y expandir su ordenador CPC. Durante la preparación del libro se ha construido, al menos, uno prototipo de cada proyecto. Por lo tanto los esquemas de los circuitos representan la forma final. Vea la página iii para los detalles de los componentes disponibles.

Los lectores que no están familiarizados con el 'hardware' de los ordenadores y su terminología, deberán leer urgentemente el Apéndice 6. Su título es "¡Empiece aquí!", y les dará una cierta experiencia en los temas que se tratan en este capítulo.

LA FUENTE DE ALIMENTACION

La fuente de alimentación del CPC produce una salida de +5 Voltios solamente. Esto se debe a que el CPC utiliza circuitos integrados que requieren solamente una alimentación de +5 Voltios CC. Hay algunos circuitos integrados que se usan en los proyectos que vamos a describir, que necesitan +5 Voltios y también +12 Voltios CC. Algunos necesitan también -12 Voltios. Debido a esto, es necesario proporcionar una fuente de alimentación externa que nos de las siguientes tensiones:

+5 voltios a 3 amperios
+12 voltios a 1 amperio
-12 voltios a 1 amperio

Los +5 voltios los proporcionamos para evitar el riesgo de sobrecargar la fuente de alimentación interna del CPC. Si no se siente capaz de construir la fuente de alimentación externa, existen varios modelos comerciales que se ajustan a nuestras especificaciones. Los cálculos de los amperios son solamente nominales por lo que puede valer cualquier unidad capaz de proporcionar algo aproximado a lo descrito.

Las figuras 3.1 y 3.2 muestran el esquema del circuitc de la fuente de alimentación externa.

La unidad está diseñada de forma muy convencional. Las tensiones principales las proporciona el transformador T1, que tiene varias salidas. Esto significa que tiene varios bobinados secundarios para producir diferentes tensiones de CA (Corriente Alterna). Este tipo de transformador es sumamente útil cuando se diseñan fuentes de alimentación de las que se quieren obtener diferentes tensiones de CC.

Cada una de las tensiones de CA es rectificada por un puente rectificador BR1 a BR4, para obtener una tensión de 'casi' CC. Decimos 'casi' porque en este punto todavía no está regulada y contiene cierta componente alterna. Después se filtran estas tensiones por medio de los condensadores C1 a C6. Estos condensadores almacenan una carga de forma que cualquier alternancia en la tensión que sale del rectificador, se puede compensar con la carga acumulada en los condensado-

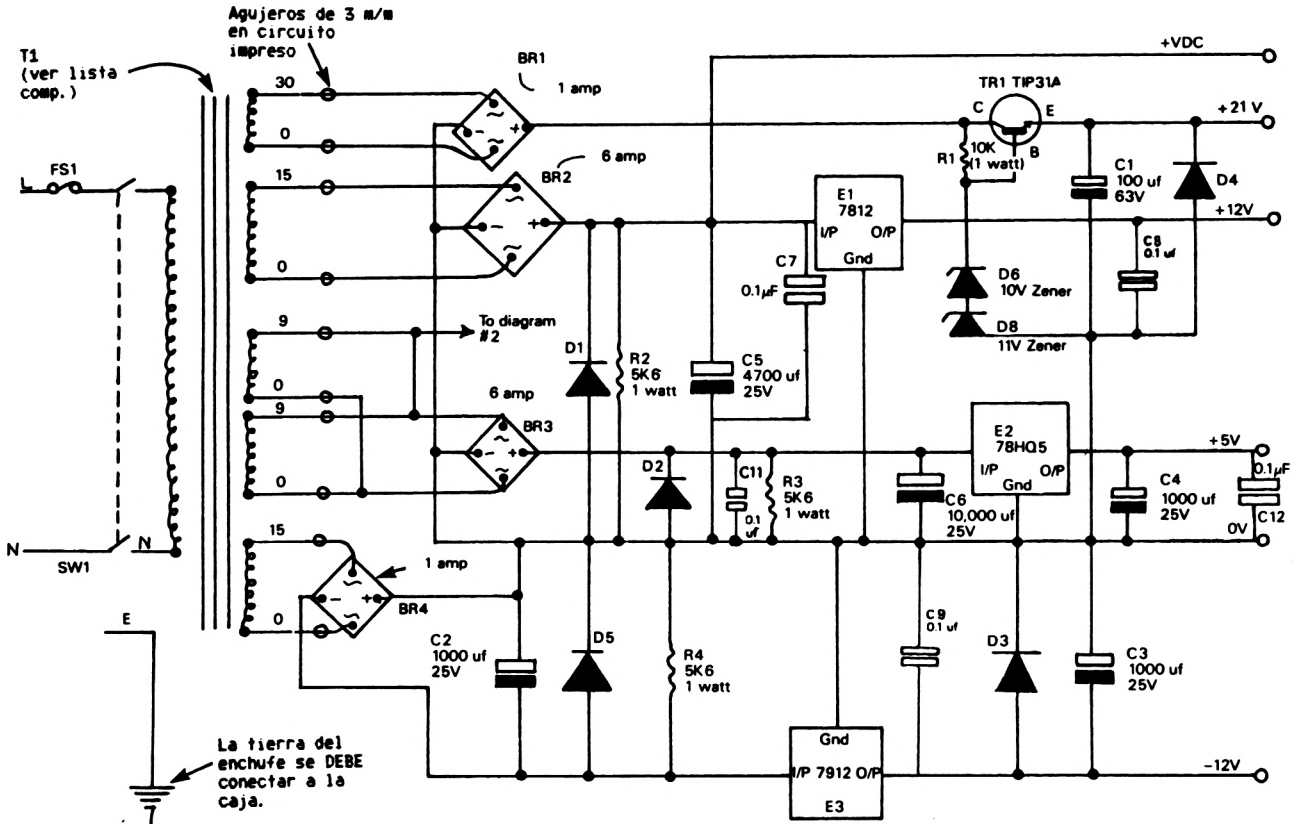


Fig.3.1 Fuente de alimentación SIGMA (Diagrama #1)

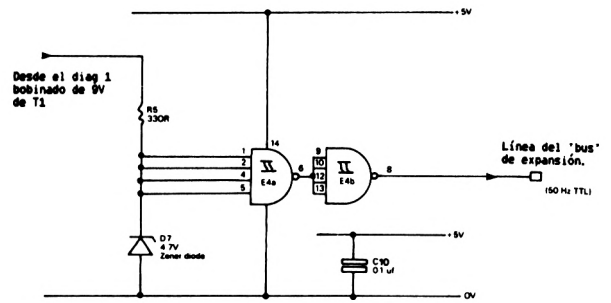


Fig. 3.2 Fuente de alimentación, diagrama #2

res. Los condensadores funcionan de diferente forma que las baterías, pueden contener suficiente carga como para cubrir la demanda por unos pocos milisegundos solamente. Las tensiones filtradas van a tres reguladores de tensión.

La función de un regulador es asegurar un suministro constante de tensión, sin importar si la carga de salida es alta o baja. En los circuitos digitales el consumo de potencia puede fluctuar bruscamente, dependiendo de la actividad que esté teniendo lugar. Esto hace esencial que las fuentes de alimentación de los ordenadores estén bien reguladas. Las salidas de los reguladores son filtradas de nuevo, y los diodos D1-D5 están colocados para evitar que las tres salidas de tensión se vayan al potencial opuesto, durante el encendido o apagado. Los reguladores se suelen cuidar ellos mismos. Están protegidos internamente contra cortocircuitos o sobrecalentamientos. Por lo tanto, si se recalientan, o si se hace un cortocircuito en su salida, se cortan automáticamente. (No intente poner a prueba la protección contra cortocircuitos, creará una gran cantidad de picos espúreos de tensión en las líneas de alimentación que podrían dañar los circuitos que estuvieran conectados a ellas. Esta protección es solamente para caso de accidente). Las resistencias R2, R3 y R4 sirven como drenaje de los condensadores de filtro, de forma que a los pocos segundos de apagar la fuente de alimentación, se hayan descargado los condensadores.

El regulador de 21 voltios formado por R1, T1, D6 y D8 y filtrado por C1 es un regulador simple. Los dos diodos zener mantienen la base de TR1 cerca de los 21 voltios y por lo tanto, el emisor no puede subir de los 21.6 voltios - muy cerca de lo que necesitamos. En la práctica la tensión suministrada es de unos 21.5 voltios con carga conectada a su salida.

SW1 es el interruptor principal, y FS1 protege al transformador contra la sobrecarga - debe ser de 1 amperio.

Ya se ha dicho , sin embargo, que ninguno de los componentes es crítico, los equivalentes pueden funcionar perfectamente bien. El transformador que se ha usado tiene dos de sus secundarios conectados en paralelo para que den la corriente necesaria. Coloque la unidad dentro de una caja metálica, y no se olvide poner la masa del enchufe (si la tiene) conectada al chasis de la caja. Esto le dará una gran

seguridad. El fusible FS1 no debe omitirse, aunque si se puede eliminar el interruptor SW1.

Condensadores

C1	100 mfd 6.3 V
C2-C4	1000 mfd 25 V
C5	4700 mfd 25 V
C6	10000 mfd 40 V
C7-C12	0.1 mfd 25 V

Resistencias

R1	10K Ohm 1 W
R2-R4	5.6K Ohm 1 W
R5	330 Ohm

Semiconductores

D1-D5	Diodos 400V 3 Amp.
D6	Diodo Zener 10 V
D7	Diodo Zener 11 V
D8	Diodo Zener 4.7 V
BR1 y BR4	Puente rectificador de 1 Amp
BR2 y BR3	Puente rectificador de 6 Amp
E1	Regulador 1 Amp +12 V (7812)
E2	Regulador 5 Amp +5 V (78H05)
E3	Regulador 1 Amp -12 V (7912)
E4	Chip TTL 7413
TR1	Transistor TIP31A

Transformador

T1	Transformador especial con secundarios de: 30,15,9+9,15 Voltios

Varios

FS1	Portafusibles para circuito impreso
Fusible	Fusible rápido de 1 Amp 20mm
Circuito impreso	
Soldador	
Cable.	

Figura 3.3

PROYECTO 1: INTERFASE RS232 - VERSIÓN "A"

El primer proyecto es relativamente simple, convierte los datos que salen por la puerta paralela del CPC, en una secuencia de datos serie para poder usar impresoras ASCII que usan este sistema de recepción de datos, VDU (Unidades de Representación Visual) u otros dispositivos preparados para recibir los datos en forma serializada. Esta versión es únicamente de salida de datos; más adelante veremos otra versión del interfase RS232 que proporciona comunicación en los dos sentidos. Si lo que usted quiere es una simple representación serie de los datos que aparecen en la puerta de impresora del CPC, use este proyecto - la versión "A" del interfase RS232. Aquí tenemos las especificaciones de esta versión:

1) Toma los datos de la puerta de impresora del CPC y los convierte en datos serializados en una de las siguientes velocidades (se mide en baudios, donde un baudio viene a ser 1 ciclo por segundo):

75 baudios
150 baudios
300 baudios
600 baudios
1200 baudios
2400 baudios
4800 baudios
9600 baudios

2) Los datos serie salen con los niveles necesarios para manejar cualquier dispositivo serie (-12 a +12). No proporciona señales para el 'modem' (DTR, DSR, CD), el usuario debe obtenerlas de los niveles lógicos, si las necesita.

3) Responde a las señales de control XOFF (ASCII DC3, HEX 13) y XOFF (ASCII DC1, HEX 11) - ver más abajo.

4) La versión estándar obtiene la frecuencia de transmisión, del reloj maestro del CPC. (Se describe una sección adicional del circuito para el caso de que estas no sea satisfactorias).

5) Se suministra una indicación visual del estado de la señal de "ocupado" (busy). Esta señal indica cuando está preparado el interfase serie para obtener otro octeto de datos desde CPC. El indicador es un LED (diodo emisor de luz) que se ilumina para indicar que el interfase serie le está indicando al CPC que está ocupado.

Si conoce la utilidad de las señales XOFF y XON que se han mencionado, sáltese los dos siguientes párrafos.

Cuando una impresora recibe datos desde un ordenador, ya sea serie o de otra forma, la velocidad a que se reciben los datos puede sobrepasar a la velocidad a la que la impresora puede imprimir los caracteres. Para aliviar parcialmente este problema, muchos fabricantes proporcionan una memoria intermedia interna. Esta consiste en una memoria local que actúa en forma de primer en llegar, primero en salir. La figura 3.4 nos muestra la forma en que trabaja. Cuando viene del ordenador un nuevo carácter, se coloca al principio de la cola. El contenido de la cola se desplaza continuamente hacia arriba, el carácter de la cabeza de la cola en un momento dado es siempre el próximo a imprimir. La figura 3.5 nos muestra el estado del contenido de la cola después de que el carácter que ha entrado en la figura 3.4, ha alcanzado la cabeza de la cola. La misma impresora libera suficiente espacio como para permitir que los caracteres lleguen a la parte superior de la cola.

Esta es una representación simplificada, pero correcta, de la memoria intermedia de una impresora. El tamaño común de esta memoria es de 2048 octetos. (Abreviado normalmente por el fabricante como 2Kb). Para pequeñas cadenas de caracteres esto será suficientemente amplio, pero ¿Qué sucede con cantidades mayores de datos, que hagan que se llene la memoria intermedia?. Le debe decir al ordenador que pare de

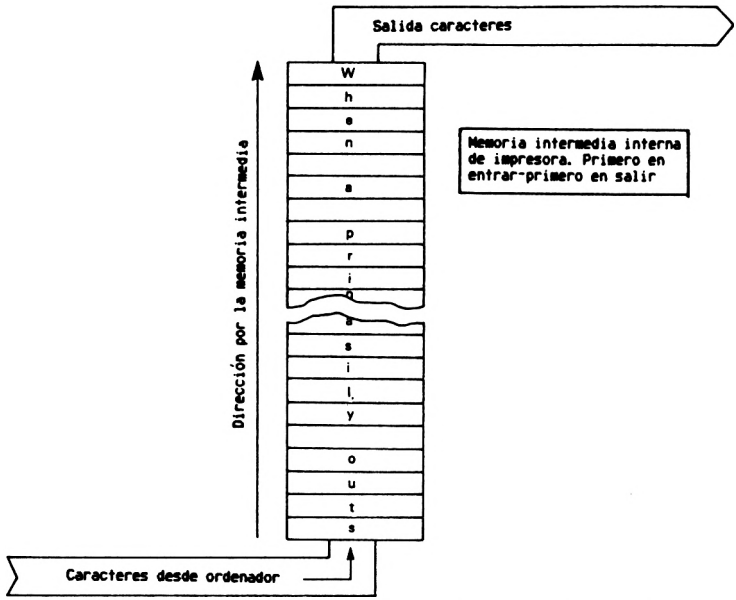


Fig. 3.4 Diagrama de la memoria intermedia de impresora (1 de 2)

enviar caracteres, o se perderán los datos. A este método se le llama XOFF/XON.

Cuando la impresora quiere decirle al ordenador que cese de mandarle caracteres, le manda un carácter XOFF (Hex 13) al ordenador. Cuando está disponible para aceptar más caracteres, una vez que ha limpiado o reducido el contenido de la memoria intermedia, la impresora le manda al ordenador el carácter XON (Hex 11), para decirle que continúe mandándole caracteres. Si la impresora tiene conectado un teclado, puede generar el carácter XOFF tecleando CTRL/S, y XON tecleando CTRL/Q. Algunas unidades de representación visual usan también el protocolo XOFF/XON cuando funcionan a grandes velocidades, como 9600 baudios. Tenga en cuenta que no todas las impresoras tienen memoria intermedia; en este caso transmiten el XOFF cuando reciben un carácter, y XON cuando lo han impreso.

Ahora pasaremos ya a los esquemas del circuito de la versión "A" del interfase RS232. (Si le confunde la terminología que se emplea para la trasmisión serie, lea la breve descripción que se proporciona en el apéndice 6). Las figuras 3.6 y 3.7 nos muestran el esquema electrónico del interfase. La figura 3.6 nos muestra el chip UART (Universal Asynchronous Receiver Transmitter - Transmisor Receptor Universal Asíncrono). Nos muestra también las diversas puertas, y los circuitos de trasmisión y recepción de datos. El CPC manda un carácter a su puerta de impresora, después pone brevemente a cero la línea de salida del sincronismo (strobe). Esto indica al mundo exterior que los datos que hay disponibles en la puerta representan un carácter para imprimir. En el caso de este circuito, la línea de sincronismo transfiere los datos al registro de trasmisión del UART. Este regis-

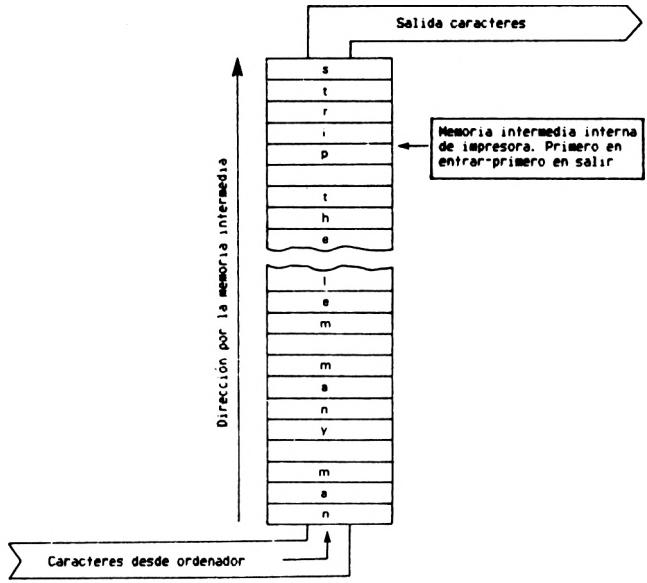


Fig. 3.5 Diagrama de la memoria intermedia de impresora (2 de 2)

tro se usa para contener el carácter que se está serializando. El UART comienza a trasmitir automáticamente el carácter, y simultáneamente se pone a cero su línea TBMT. Cuando está a cero la señal TBMT (Transmitter Buffer Empty - Memoria de Trasmisión Llena), indica que el UART no puede aceptar otro carácter. Cuando TBMT se pone a uno, el UART está disponible para aceptar otro carácter. La señal TBMT se manda al CPC como línea de "ocupado" (busy). Cuando se pone a uno la señal de "ocupado" del CPC, el programa de control no manda más caracteres a la puerta de impresora, hasta que ve que esta línea se ha puesto a nivel lógico cero. Por lo tanto la señal TBMT se ha invertido para obtener los niveles lógicos adecuados.

La puertas E3, E4 y E2d forman dos detectores. Uno de ellos detecta cuando se ha recibido el carácter XOFF desde la impresora, y el otro detecta la recepción de un carácter XON. Esto produce dos señales activas cuando están a cero, que se llaman XOFF negativa y XON negativa. Estas dos señales controlan la báscula biestable formada por las puertas E2a y E2b. Esta báscula se usa para recordar qué señal se ha recibido la última desde la impresora, XOFF o XON. Los caracteres recibidos que no sean XOFF o XON no tienen efecto. Por lo tanto, el nivel lógico de la patilla 3 de la puerta E2a indica cuál es la que se ha recibido la última. Si la patilla 3 está a uno, se ha recibido XON, mientras que si está a cero, ha sido XOFF.

En el esquema completo del circuito, la señal "ocupado" se toma en su condición verdadera, si TBMT es uno lógico, o si XOFF ha sido el último carácter que se ha recibido de la impresora. Como vimos anteriormente la señal de "ocupado" se pone a uno lógico para prevenir que el CPC mande más datos a la puerta de impresora.

En la figura 3.6 se ve el chip 1488 de trasmisión RS232. Este toma los niveles lógicos estándar y los cambia a los niveles correspondientes -12 V y +12 V. Esto aumenta la inmunidad de la señal a los ruidos en el cable que lleva los datos a la impresora. (Puede trabajar perfectamente hasta con 15 metros). Esto le permite también tener la impresora separada del ordenador y, más importante, alejar el ruido que produce. No he querido añadir el chip 1489, que realiza la función opuesta al 1488, ya que se usaría solo la cuarta parte. En su lugar he puesto dos diodos, D1 y D2, y una resistencia, R1, que limitan la tensión de entrada a +5.6 V, y -0.6 V, y que junto al transistor TR1 forman el receptor de datos. Esto resulta aceptable, ya que los chips estándar de trasmisión usados en la mayoría de los terminales e impresoras, tiene limitador de corriente. El transistor TR1 invierte los datos de entrada - un trabajo que suele hacerse con el chip 1489. Estos datos invertidos pasan a la patilla 20 del UART, que es la entrada serie.

Examinemos ahora la función del puente LK1. (Y para esta conexión todos los lectores deben leer el apéndice 7 después de terminar esta sección). Aclaremos también otro punto importante acerca de la puerta de impresora del CPC. LK1 le permite usar una impresora o una unidad de representación visual que use códigos de solo siete bits, con el octavo bit como bit de paridad, o puesto siempre a uno. Si usa un terminal o impresora de solo siete bits, se debe poner el puente LK1 desde la patilla a 0 V. si usa un dispositivo de ocho bits, LK1 se debe conectar entre la patilla 5 de la puerta E3b a la patilla 5 de E1. Aunque no hay ninguna referencia en el manual, la puerta de impresora del CPC es de siete bits. Consultando el manual técnico se puede ver que el bit siete está conectado a nivel lógico cero dentro de la misma máquina, así que no importa los datos que se manden a la impresora, el bit superior (que es el de orden más alto), se recibirá siempre como un cero. Debido a esto, la trasferencia de cosas como código máquina por medio de la puerta de impresora no se puede llevar a cabo satisfactoriamente.

En la figura 3.6 se puede ver que hay muchas patillas del UART que se han conectado a 0V o a +5V. La mayoría de ellas se usan para la selección de las características de los datos serie. Usando las conexiones como se han puesto, los datos serie se mandarán con:

8 bits de datos
2 bits de parada
Sin comprobación de paridad

Si precisa cambiar estas características, lea el apéndice A donde se reproduce la hoja de características del UART, detallando todas las opciones disponibles y cómo seleccionarlas.

Finalmente en la figura 3.6, se ven las dos señales de reloj, entrando por la parte superior derecha del esquema - llamadas TXC y RXC - son los impulsos de reloj para las secciones de trasmisión y recepción del UART. Las velocidades de trasmisión y recepción de datos se deciden por medio de la frecuencia de la señal que se introduce por estas dos entradas. Para complicar un poco más las cosas, el UART divide la frecuencia, que es 16 veces la de trasmisión. (Como veremos un poco más adelante en la versión "B" del interfase RS232, otros chips receptores/transmisores son más programables).

Veamos ahora la figura 3.7, las señales de reloj se pueden obtener a partir del reloj maestro del CPC, que funciona a una frecuencia de

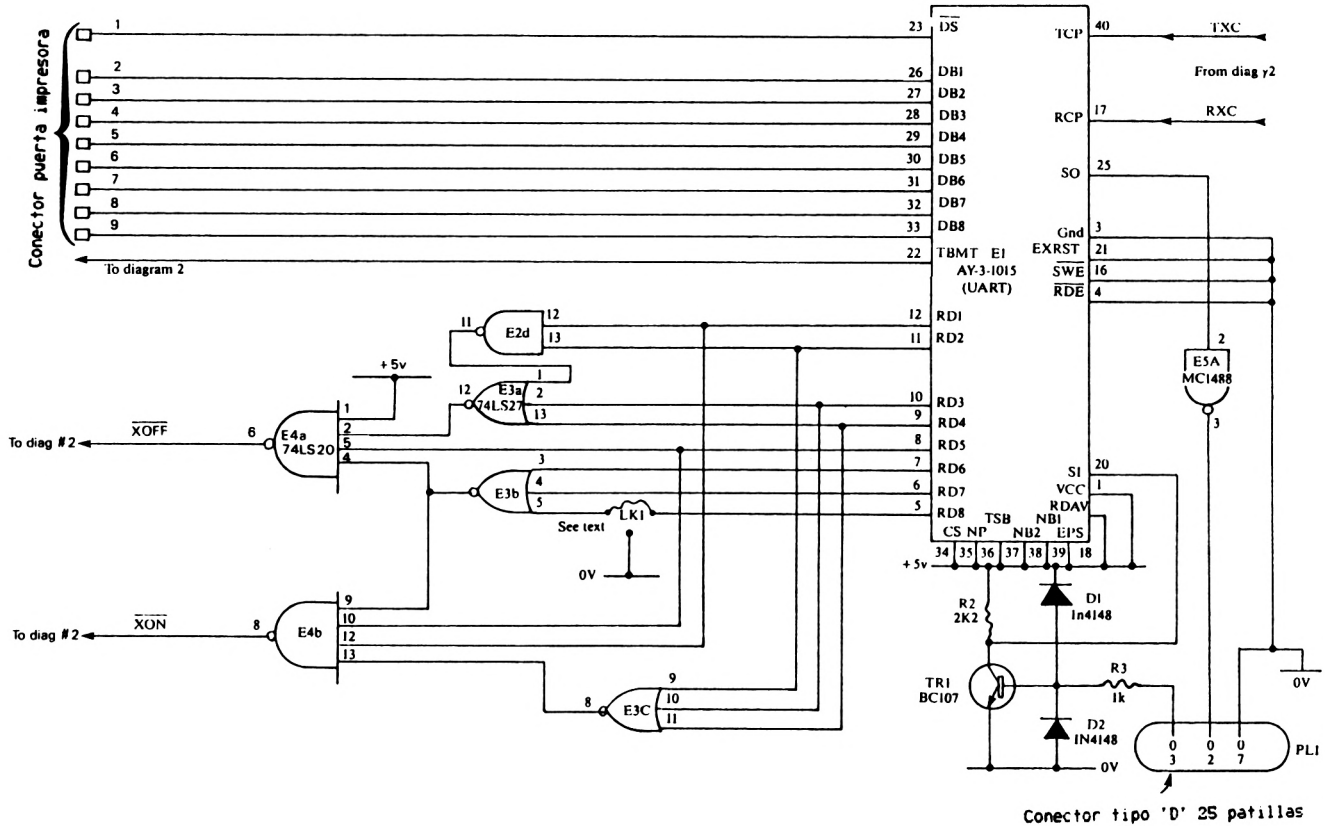


Fig.3.6 RS232 versión "A", Diagrama #1

4 Megaciclos, (4 millones de ciclos por segundo). Si va a usar un circuito estándar, y piensa que puede manejar aceptablemente un pequeño soldador, le sugerimos una modificación que puede hacer solamente al CPC464, que le evitará tener que usar dos conectores, uno en la puerta de la impresora y el otro en el "bus" de expansión del CPC, cuando use la versión "A" del interfase RS232. La función de este cambio es proporcionar la señal de reloj de 4 Megaciclos en la puerta de la impresora. TENGA EN CUENTA que si realiza esta transformación invalidará la garantía. Si no la quiere realizar, también puede usar la versión "A" del interfase RS232, pero tendrá que usar un conector de 50 vías para el "bus" de expansión, para obtener la señal de 4 Megaciclos de la línea 50.

Asegúrese de que los cables del monitor y del teclado están desconectados antes de comenzar.

La modificación consiste en los siguientes pasos. Primero, desconecte el circuito impreso principal del teclado del CPC464. Para hacerlo, quite los seis tornillos de la base de la unidad del teclado. Con mucho cuidado quite los conectores que unen la unidad de cinta al circuito impreso principal. Suelde cuidadosamente un cable fino desde la patilla número 17 de la puerta de impresora. (Use el diagrama del apéndice cinco, página 2 del manual del usuario de CPC464, para localizar la patilla 17).Después, corte el cable a la longitud necesaria para que llegue al final de la resistencia R141 que hay cerca al conector de 50 vías de la puerta de expansión. La figura 3.9 nos muestra los detalles de esta conexión vista por la parte de los componentes de la tarjeta principal del CPC464. Asegúrese cuidadosamente de que la soldadura no pase a la otra parte del conector, especialmente la parte que forma el contacto. Asegúrese de no echar salpicaduras de soldadura sobre el circuito. Ahora ya ha completado la modificación, reensamble la unidad en orden inverso al que usó para desmontarla.

En la versión estándar (como se ve en la figura 3.7) la señal de 4 Megaciclos del CPC se introduce desde un chip divisor a otro. Cada uno de estos son chips 4040 CMOS contadores divisores por dos de 12 pasos. El primer divisor tiene una longitud de cuenta de 13 (HEX D) - detectados por la puerta E8a. Cuando llega el 13, E8a restaura el primer divisor, y suma uno al segundo contador divisor. Así, introduciendo 4 MHz en E6, y, después de dividirlo, obtenemos una frecuencia de 307692.3 Hz. Esta es dividida por dos en pasos sucesivos del divisor E7, de forma que en la patilla 9 de E7 obtenemos una frecuencia de 153846 Hz - que cuando la divide el UART por 16 nos da 9616 Hz redondeando. Esta es suficientemente parecida a los 9600 Hz que necesitamos.

He probado el interfase RS232 versión "A" con diferentes impresoras y terminales, y a 9600 y 600 baudios no he obtenido ningún error en los caracteres trasmitidos. Por lo tanto no tengo reserva en decir que en casi todas las aplicaciones con velocidades estándar de trasmisión, funcionará correctamente el circuito de reloj de la figura 3.7. Si se le presenta algún problema cuando lo intente usted, puede recurrir al montaje que se ve en la figura 3.8 (Que será el sistema estándar, a menos que se use el otro más barato). En este montaje se usa el chip de Ferranti F-4702-BPC generador de velocidades de trasmisión, y la selección de baudios se realiza por medio de un código de cuatro bits a la entrada del chip. El circuito de la figura 3.8 se usa en lugar de todos los componentes que aparecen dentro del recuadro punteado de la figura 3.7.

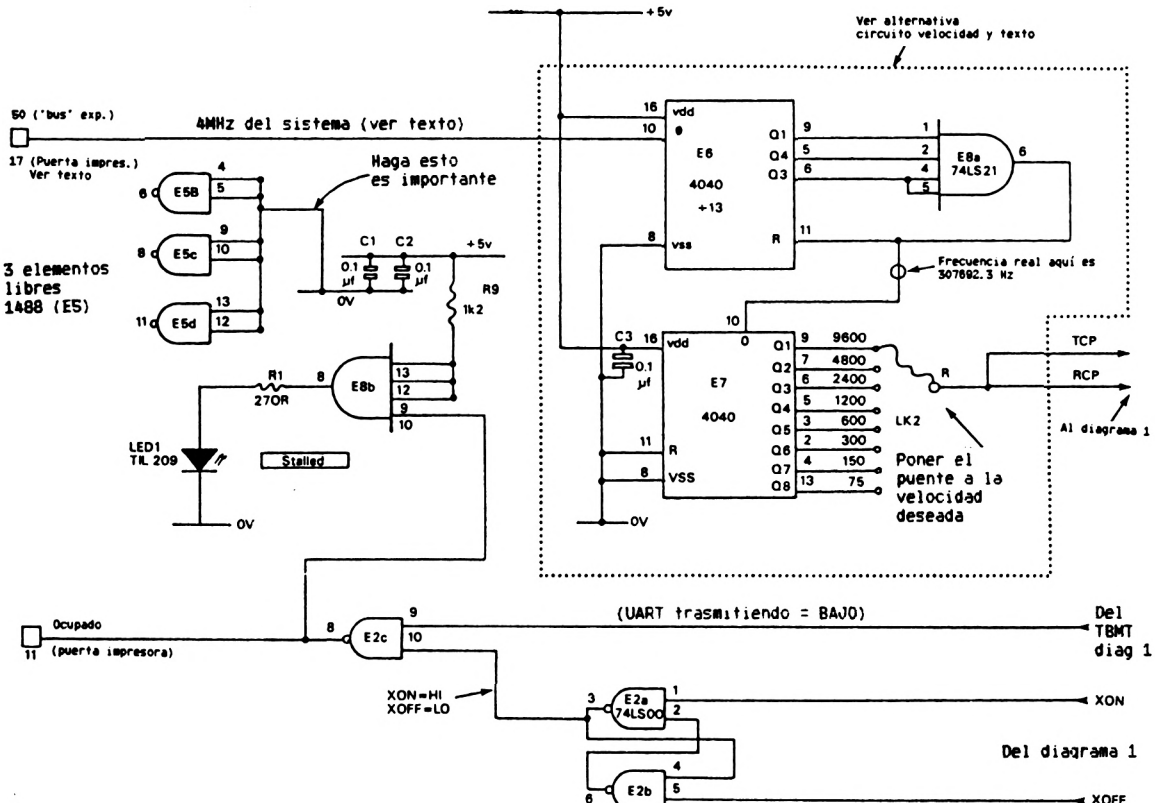


Fig.3.7 RS232 versión "A", Diagrama #2

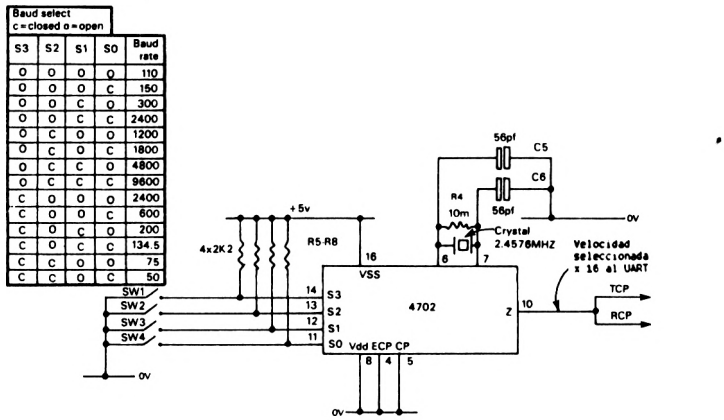


Fig. 3.8 Circuito generador alternativo de velocidad de trasmisión

Usando el esquema estándar para la derivación de la frecuencia de 9600 baudios, el resto de las frecuencias se obtienen a través de todos los pasos del segundo divisor - E7. Ya que todas las frecuencias están disponibles simultáneamente usando este esquema, se puede conectar un puente desde la salida requerida de E7 a las entradas TCP y RCP del UART. (Patillas 40 y 17 respectivamente).

El resto del circuito de la figura 3.7 consiste en: E8b que se usa para manejar el LED. Este diodo da una indicación visual del estado de la línea "ocupado" para el CPC. Cuando el LED está encendido, le está diciendo al ordenador que no mande datos; cuando el LED está apagado, el ordenador es libre de mandar un carácter. Es MUY IMPORTANTE unir todas las entradas no usadas de E5 a cero voltios. Si no se hace esto, el chip puede calentarse excesivamente debido al ruido que se presenta en las entradas, haciendo que los circuitos sobrantes sean inestables. Una vez vi un caso extremo en el que un chip 1488 se perforó por el calor; recuerde, hay un potencial de 24 voltios en el chip. La puerta E2c pone la señal "ocupado" como verdadera si se manda XOFF desde la impresora, o si TBMT está a cero.

Cuando haya terminado este interfase RS232 necesitará comprobar que funciona correctamente. El método más popular de comprobarlo es sacar a la impresora un serie de caracteres ASCII. Este método consiste en imprimir todos los caracteres ASCII de forma rotativa. (Vea el apéndice cuatro para el juego de caracteres ASCII). Esto asegura que todos los caracteres pueden imprimirse en todas las posiciones de cada línea, y tiene la ventaja adicional de que los principales problemas con la trasmisión de datos a una impresora, se ven como roturas en la representación regular en la impresora. La figura 3.10 muestra un programa para el CPC que imprimirá este tipo de prueba, junto con un ejemplo de la impresión que se debe obtener.

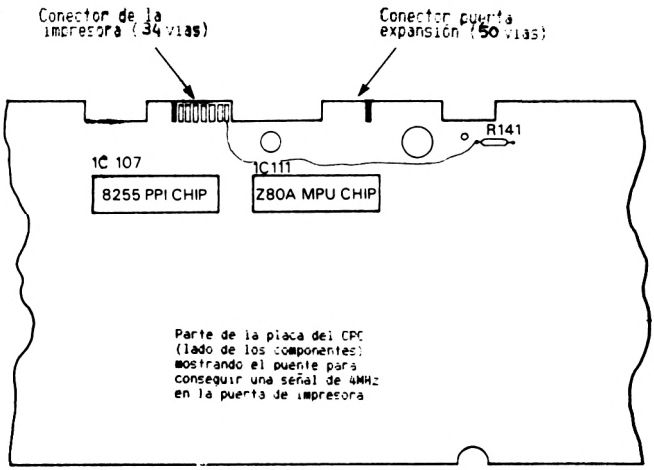


Fig 3.9 Modificación para obtener los 4 MHz

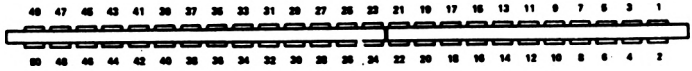
```
50    '** PRUEBA DE LA IMPRESORA CON 60 LINEAS DE CARACTERES ASCII **
100   MODE 2: STRM=8 'Prepara el modo de la pantalla y pone la impresora en
      el canal 8
105   INPUT "Columnas de impresión <80>";W: IF W > 132 THEN 105 ELSE
      IF W=0 THEN W=80
110   FOR I=32 TO 32+79: C$=C$+CHR$(I): NEXT I:
      IF LEN(C$) < W THEN 110
115   C$=LEFT$(C$,W): CLS
120   PRINT "IMPRIMIENDO...";: LOCATE 16,1: PRINT "60 líneas"
130   FOR I=1 TO 60: LOCATE 12,1: PRINT I;: PRINT #STRM,C$
140   C$=MID$(C$,2,LEN(C$))+LEFT$(C$,1): NEXT
10000 END
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmn
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !
'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"
()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#
)*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$
*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%
+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&
,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&'
-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&'(
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&'()
/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&'()*
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno !"#$%&'()*+
```

Fig. 3.10 Programa de impresión de prueba ASCII

Zócalo de expansión

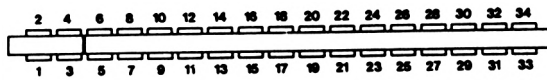
VISTO POR DETRAS



Patilla 1 SOUND	Patilla 18 A0	Patilla 35 INT
Patilla 2 GND	Patilla 19 D7	Patilla 36 NMI
Patilla 3 A15	Patilla 20 D6	Patilla 37 BUSR2
Patilla 4 A14	Patilla 21 D5	Patilla 38 BUSAK
Patilla 5 A13	Patilla 22 D4	Patilla 39 READY
Patilla 6 A12	Patilla 23 D3	Patilla 40 BUS RESET
Patilla 7 A11	Patilla 24 D2	Patilla 41 RESET
Patilla 8 A10	Patilla 25 D1	Patilla 42 ROMEN
Patilla 9 A9	Patilla 26 D0	Patilla 43 ROMDIS
Patilla 10 A8	Patilla 27 +5v	Patilla 44 RAMRD
Patilla 11 A7	Patilla 28 MREQ	Patilla 45 RAMDIS
Patilla 12 A6	Patilla 29 MI	Patilla 46 CURSOR
Patilla 13 A5	Patilla 30 RFSH	Patilla 47 L. PEN
Patilla 14 A4	Patilla 31 IORQ	Patilla 48 EXP
Patilla 15 A3	Patilla 32 RD	Patilla 49 GND
Patilla 16 A2	Patilla 33 WR	Patilla 50
Patilla 17 A1	Patilla 34 HALT	

Zócalo para la segunda unidad de disco

VISTO POR DETRAS



Patilla 1 READY	Patilla 18 GND
Patilla 2 GND	Patilla 19 MOTOR ON
Patilla 3 SIDE 1 SELECT	Patilla 20 GND
Patilla 4 GND	Patilla 21 N/C
Patilla 5 READ DATA	Patilla 22 GND
Patilla 6 GND	Patilla 23 DRIVE SELECT 1
Patilla 7 WRITE PROTECT	Patilla 24 GND
Patilla 8 GND	Patilla 25 N/C
Patilla 9 TRACK 0	Patilla 26 GND
Patilla 10 GND	Patilla 27 INDEX
Patilla 11 WRITE GATE	Patilla 28 GND
Patilla 12 GND	Patilla 29 N/C
Patilla 13 WRITE DATA	Patilla 30 GND
Patilla 14 GND	Patilla 31 N/C
Patilla 15 STEP	Patilla 32 GND
Patilla 16 GND	Patilla 33 N/C
Patilla 17 DIRECTION SELECT	Patilla 34 GND

El método exacto de conectar la fuente de alimentación se dejará a gusto del lector, ya que depende realmente de la forma en que haya terminado el proyecto, y también de los proyectos de este libro que vaya a construir.

Finalmente, se muestran en la figura 3.11 las patillas de alimentación de los chips que se han usado.

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	OV
1	AY-3-1015	UART	1	--	--	3
2	7400	4 puertas NAND 2 entradas	14	--	--	7
3	74LS27	3 puertas NOR 3 entradas	14	--	--	7
4	74LS20	2 puertas NAND 4 entradas	14	--	--	7
5	MC1488 ó	Controlador RS232	--	14	1	7
	SN75188	" "	--	14	1	7
6	40408	Divisor de 12 pasos	16	--	--	8
7	40408	Divisor de 12 pasos	16	--	--	8
8	74LS21	2 puertas AND 4 entradas	14	--	--	7
9	4702BPC	Generador vel. trasmisión	16	--	--	8

Fig. 3.11 Patillas de alimentación de los chips usados

Lista de componentes para el interfase RS232 versión "A"

Condensadores

C1-C3 0.1 mfd 16 volt disco cerámicos
C5-C6 * 56 pf 16 volt poliestireno

Resistencias

R1 270 Ohm 1/2 watio
R2 2K2 Ohm 1/4 watio
R3 1K Ohm 1/4 watio
R4 * 10M Ohm 1/2 watio
R5-R8 * 2K2 Ohm 1/4 watio
R9 1K2 Ohm 1/4 watio

Semiconductores

E1 AY-3-1015 UART chip
E2 TTL 74LS00 4 puertas NAND dos entradas
E3 TTL 74LS27 3 puertas NOR tres entradas
E4 TTL 74LS20 2 puertas NAND cuatro entradas
E5 MC 1488 chip controlador de línea
E6 y E7 CMOS 4040 divisor de 12 pasos
E8 TTL 74LS21 2 puertas AND cuatro entradas
E9 * 4702 generador de velocidades de trasmisión
Cristal * 2.4576 Mhz
LED1 LED de 5 mm
TR1 Transistor BC107
D1 y D2 Diodos 1N4148 silicona

Varios

Zócalos para todos los chips
Circuito impreso
SW1-SW4 * Conmutadores de selección de baudios para el generador de velocidad de trasmisión (conmutador DIL de 4 vías)
PL2 Conector CANON hembra de 25 vías

Cable de 34 conductores y conector IDC para conectar a la puerta de impresora del CPC

Una caja de plástico adecuada

Fig. 3.12 Lista de componentes para el interfase RS232 versión "A"

TARJETA MULTI PROYECTO (PROYECTOS 2 A 6)

El sintetizador de palabra y los cuatro proyectos que le siguen están diseñados para ser construidos sobre una sola tarjeta. Esto se hace así porque los cinco proyectos hacen uso - de una forma u otra - de los dos chips PPI de la tarjeta. (Vea el capítulo uno y el apéndice tres para los detalles del chip PPI). Incluyéndolos todos en una sola tarjeta, reducimos la cantidad de decodificadores de direcciones requeridos, siendo necesario un solo decodificador. El diagrama de bloques de la tarjeta multi-proyecto se muestra en la figura 3.13. La lista de componentes de cada proyecto se dará por separado, listando solamente aquellas partes que pueda necesitar para construir el proyecto aislado. Ya que algunos componentes se comparten entre varios proyectos, tenga cuidado de no comprar por partida doble cuando solo necesite uno. Al final de la descripción de la tarjeta multiproyecto se da una lista de los componentes.

PROYECTO DOS: SINTETIZADOR DE PALABRA

La síntesis electrónica de palabras se ha abaratado recientemente debido a su aplicación a los ordenadores caseros. Ahora es posible comprar circuitos integrados que reproducen una voz inteligible, por menos de 2500 pesetas. Hay varios tipos de sintetización electrónica de la palabra.

La primera técnica, y la más sencilla, es conectar un micrófono a un circuito conversor analógico digital. La salida resultante de este conversor A/D se almacena en gran cantidad de memoria. Cuando se reproducen las palabras almacenadas, la representación digital se pasa a un conversor Digital Analógico (D/A), a la misma velocidad a que se 'grabó'. Esto convierte los datos digitales en palabras. En los pequeños ordenadores no se usa mucho esta técnica, debido a que un segundo de conversación, almacenado de esta forma, requiere cerca de 12000 octetos de memoria. Esto puede consumir 64K de memoria del CPC en solo cinco segundos. Otra forma de almacenar palabras es dividirlas en componentes más pequeños. El componente más pequeño de una palabra se conoce como un "alófono". Se puede almacenar en un circuito integrado la información requerida para reproducir estos alófonos. Este método es bastante más práctico para los pequeños sistemas. Todo lo que tiene que hacer ahora el microprocesador, es pasar los códigos

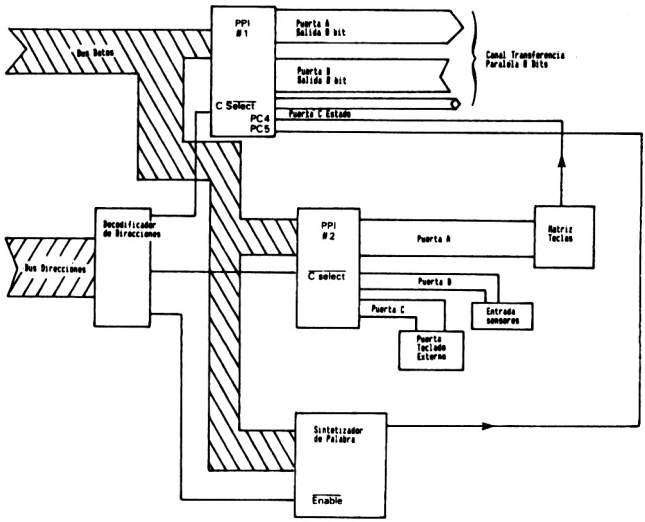


Fig. 3.13 Diagrama de bloques de la tarjeta multiproyecto

apropiados dentro del chip sintetizador, para hacerle reproducir una cadena de componentes que forman la palabra o palabras deseadas. Una tercera variante, que es bastante similar a la anterior, es almacenar información de un vocabulario limitado dentro de un chip. Este método es menos flexible que el anterior, pero la calidad de las palabras almacenadas es ligeramente superior.

Obviamente, el chip sintetizador de palabra que usted elija debe acoplarse a su aplicación. En casos en que solo hay cierto número de palabras para reproducir, se puede elegir la claridad extra de un vocabulario fijo. Si quiere tener un sintetizador de palabras que pueda emitir palabras poco usuales en el vocabulario normal, debe usar el chip que reproduce cada alófono individual bajo el control del microprocesador. Este es el método que hemos usado en nuestro proyecto de sintetizador de palabra.

El chip sintetizador que vamos a usar en este diseño es el SPO256 de General Instruments. Este chip puede reproducir una gran cantidad de alófonos útiles, y tiene la gran virtud de la simplicidad. La programación del SPO256 es análoga a mandar datos al USART. El SPO256 toma un código de 6 bits, que representa el alófono requerido, en el umbral negativo de la señal ALD. Después se reproduce el alófono, y cuando el chip está listo para recibir otro comando, se ponen a uno sus líneas LRQ y SBY. En el apéndice tres se reproduce la disposición de las patillas del SPO-256. El diagrama de bloques del circuito que estamos tratando se muestra en la figura 3.14.

La figura 3.14 muestra el sintetizador de palabra acoplado a un registro de inflexión. Esto permite a la aplicación programar el sintetizador para que cuando hable, use uno de siete tonos de voz. No hace falta decir que esto añade gran naturalidad a la calidad de la sintetización, y se aleja bastante de la voz monótona de los sinteti-

zadores que se suelen ver, u oír. Es posible controlar la inflexión gracias a que el tono de la voz reproducido por el SPO256 depende de la frecuencia presentada en su entrada OSC1. (Puede usar también el SPO256 con un cristal de 3.12 MHz - ver la hoja de características del apéndice tres). En este diseño, la salida del registro de inflexión - que es realmente una báscula dando entrada a un conversor pasivo digital a analógico - se introduce en la entrada de control de frecuencia de un chip oscilador controlado por tensión. La frecuencia de la salida de este chip se ajusta variando la tensión de su entrada de control de frecuencia. No se ha añadido un amplificador separado en el diseño. Esto se hace para simplificar y economizar, pero también porque la calidad que se obtiene introduciendo la señal de nuevo en el amplificador interno del CPC es muy aceptable para una unidad tan sencilla.

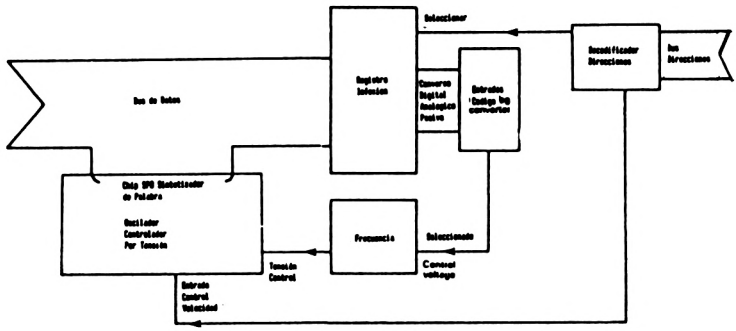
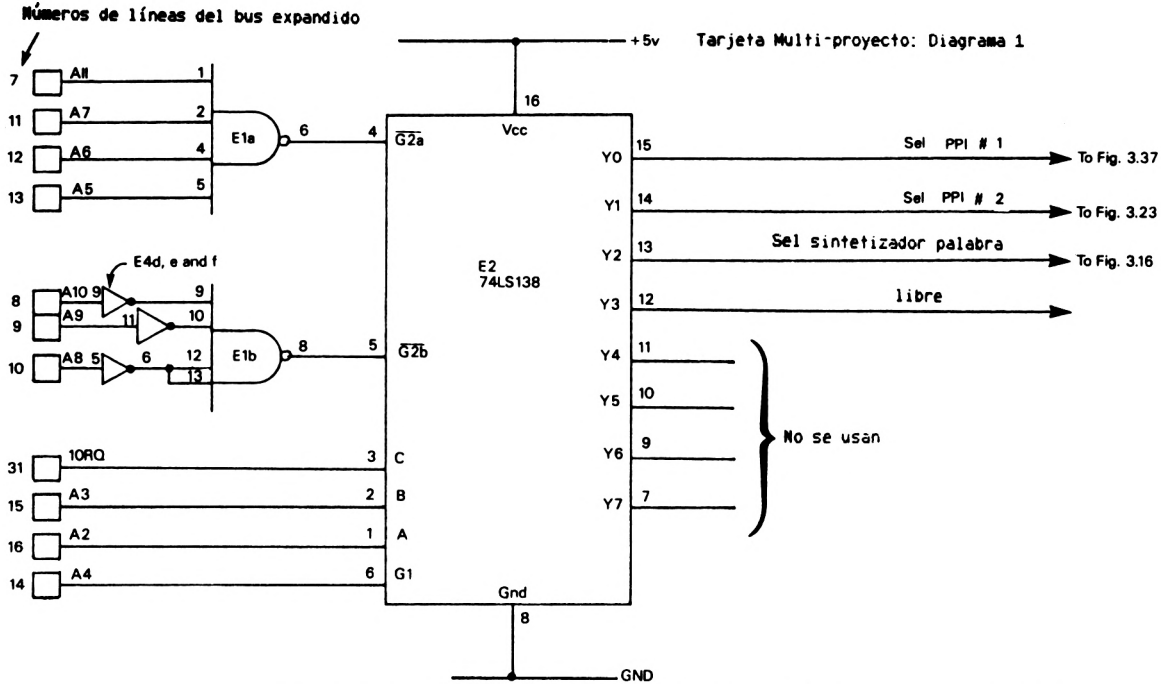


Fig. 3.14 Diagrama de bloques del sintetizador de palabra

La figura 3.15 es el esquema del decodificador de direcciones de la tarjeta multiproyecto. Es autoexplicativa, y sitúa el registro del sintetizador de palabras en la dirección de E/S Hex F8F8. Cuando el decodificador de direcciones - preparado de la forma que se dijo en el capítulo 1 - detecta que en el 'bus' de direcciones aparece la dirección de la página de E/S, que está dentro del rango F8F8 a F8FB, la salida Y2 del chip E2 (marcado en el esquema como SEL SPEECH SYNTH) se pone a cero. Situándonos un poco más adelante en nuestro sintetizador de palabras, vemos que la señal Y2 pasa a la figura 3.16 a través del grupo de puertas lógicas formado por E8a-E8d y por los inversores E4b y E4c. Estas puertas lógicas aseguran que la línea CLK que entra en el registro de inflexión, E10, se pone a uno solamente cuando la dirección de E/S Hex F8F9 está presente en el 'bus' de direcciones. Más adelante, la entrada ALD negativa del chip SPO256 se pone momentáneamente a cero durante el tiempo en que aparece una señal de escritura a la dirección de E/S Hex F8F8. (La dirección es una señal que, cuando se pone a cero, permite que pasen los datos del 'bus' al SO256). Todo esto se resume en que cuando haya construido y conectado la unidad, el comando:

```
OUT $F8F8,15
```

hará que el sintetizador de palabras tome vida. Este es el alófono cuyo código es 15, que es el sonido inglés AX, que viene a ser como el español AC, que se puede usar como parte de la palabra ACTUAL. Los comandos:



Valor Bit	X	X	X	X	1	0	0	0	1	1	1	1	0	0	0	0
Num. Lin. direccion	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Hex Valves	X (set to F)				8				F				0			

Fig 3.15 Esquema del decodificador de direcciones de la tarjeta multiproyecto

```
FOR I=0 TO 7: OUT $F8F9,I: FOR P=1 TO 1000: NEXT: NEXT
```

genera un sonido que comienza bajo y va subiendo a través de ocho tonos diferentes. Esto nos hace recordar un punto muy importante acerca del chip 'hablador'. A menos que usted le diga que se pare, seguirá produciendo el sonido del alófono que le haya dicho usted. Afortunadamente los códigos de los alófonos 0-4 están definidos como silencios de distintas longitudes, por lo que antes de que se vuelva loco teclee el comando:

```
OUT $F8F8,0
```

Este comando le hará callarse. La dirección del circuito decodificador pone el registro del chip en la dirección de E/S Hex F8F8. Sin embargo, este registro solo es de escritura, es decir, que aunque ponga en él datos, su lectura con el comando:

```
PRINT INP($F8F8)
```

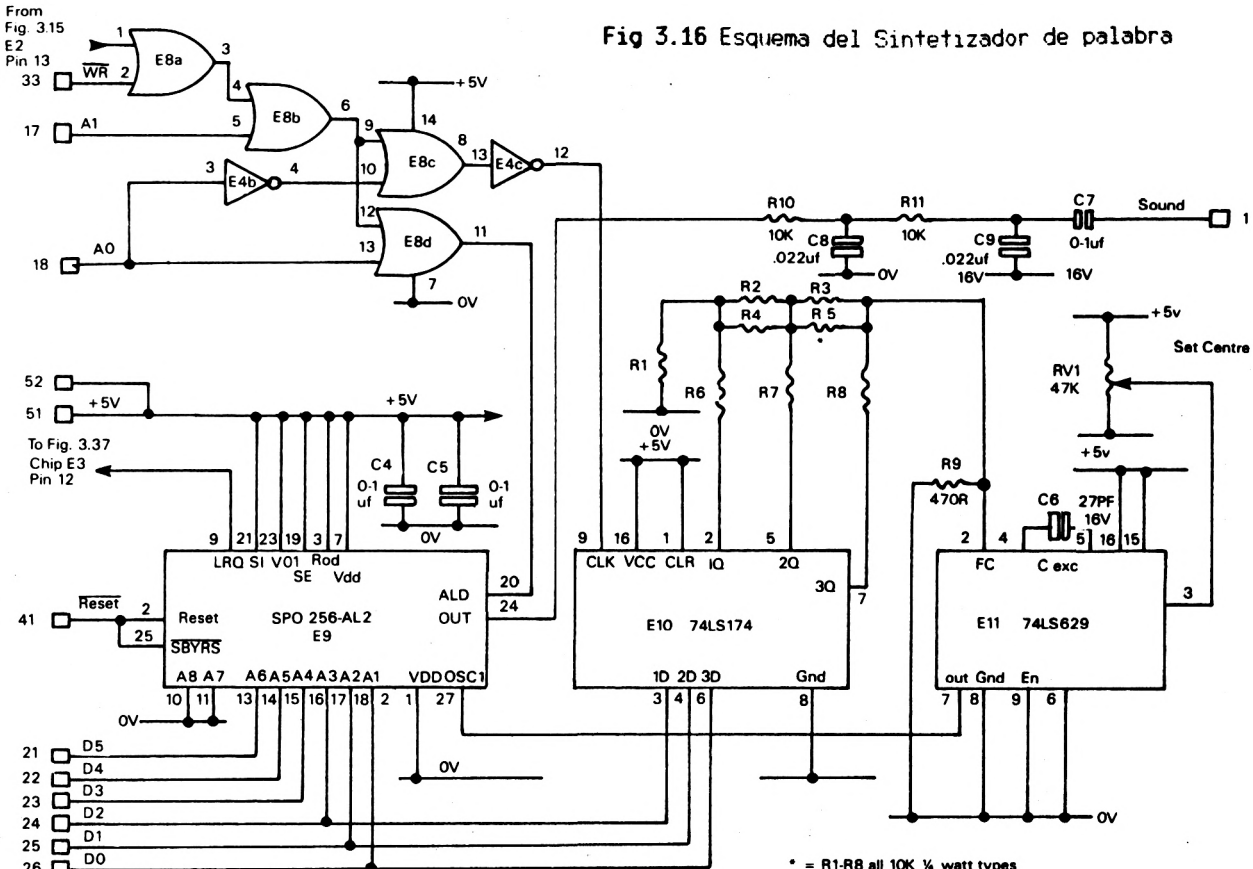
siempre nos dará "255". Esto se debe a que el chip no responde cuando el procesador lee la dirección de E/S F8F8, y el efecto es el mismo que el de leer una dirección vacía de un espacio de direcciones. Así que ¿cómo podemos leer el estado del chip?. Hay solamente dos señales de estado que nos pueden interesar en esta aplicación, se llaman LREQ negativa y SBY. Se puede usar cualquiera de estas señales para que el procesador sepa cuándo está preparado el chip para otro código de alófono. La línea LBAR negativa es una salida que se pone a cero cuando el chip está preparado para aceptar otro código, mientras que la salida SBY se pone a uno para indicar lo mismo. En este diseño se ha usado la señal LREQ negativa que se introduce en uno de los dos chips PPI - E3 patilla 12.

El chip E3 está etiquetado dentro de este diseño, como el PPI número uno (Abreviado en los diagramas como PPI #1). No queremos meternos con los chips PPI en este capítulo (ya se verán con más detenimiento en el próximo proyecto), así que baste saber que el PPI está preparado para que su patilla 12 sea la entrada al bit 5 de la puerta C del PPI uno. El programa solo tiene que leer la puerta C del PPI uno, hacer un AND del bit 5, y si está a cero es que el chip está preparado para recibir otro código de alófono. Pronto veremos un programa práctico para manejar el sintetizador de palabras, junto con una lista de los códigos de alófonos.

El chip sintetizador de palabras no está dirigido por un circuito oscilador de cristal, sino por un chip 74LS629 (Oscilador Controlado por Tensión - chip VCO). El potenciómetro RV1 ajusta la frecuencia central del oscilador y se debe poner a la mitad de su recorrido, cuando se prepare el circuito. El condensador temporizador C6 es el otro factor para la determinación de la frecuencia. La tensión aplicada a la entrada del Controlador de Frecuencia (FC) del 74LS629 aumenta o disminuye la frecuencia del oscilador. Una tensión baja lo frena, mientras que una alta le permite funcionar más rápidamente. En este diseño, el programa coloca un valor entre 0 y 7 en el registro de inflexión formado por E10, que es convertido en uno de ocho posibles niveles de tensión por la red R2R, constituida por R1-R8. El nivel de tensión seleccionado se aplica a la entrada FC del VCO, alterando la frecuencia de su salida. Como el chip sintetizador depende enteramente de la salida VCO para todos sus tiempos, alterando la frecuencia de salida del VCO alteramos la velocidad y el tono en el que habla el SP0256.

Números de líneas del 'bus' de expansión

Fig 3.16 Esquema del Sintetizador de palabra



La parte final del circuito consiste en un pequeño filtro C/R construido con R10, R11, C8 y C9. Esto hace que la salida sea más adecuada para introducirla en el amplificador interno del CPC. La señal de audio del sintetizador se introduce en la entrada SOUND negativa del 'bus' de expansión que va directamente al amplificador del cassette. En un CPC es posible usar el 'datacorder' para grabar o cargar programas mientras está hablando el sintetizador - por lo que no parece que haya interacción entre el cassette y el sintetizador.

Se necesita cierta preparación para asegurar una operación correcta del circuito sintetizador de palabras. Principalmente consiste en en poner RV1 en un punto donde el SPO256 esté coordinado con el registro de inflexión cargado con cero o siete. Si falla el sintetizador, debido a que el reloj va demasiado rápido o demasiado lento, emite un sonido ininteligible que suena como una cascada de agua. La única forma de pararlo es apagando todo y arrancando de nuevo. Si no tiene un osciloscopio disponible, el ajuste es una cuestión de pruebas sucesivas, pero suele ser suficiente un par de minutos para hacerlo funcionar en condiciones aceptables. Si tiene un osciloscopio, puede ajustar rápidamente el circuito. Ponga un valor de 3 o 4 en el registro de inflexión y mire en la patilla 27 del chip sintetizador. Ajuste RV1 de forma que vea una frecuencia de unos 3 megaciclos. El ajuste está terminado.

Notas de construcción

Aparte de las normas de seguridad contra la electricidad estática que debe usted tomar cuando maneje circuitos integrados MOS, como el SPO256, no debe tener ninguna dificultad en el ensamblaje del sintetizador de palabras - suponiendo que esté usando un circuito impreso. (vea las notas sobre el circuito impreso al principio de este libro).

La figura 13.7 es una lista de los alófonos disponibles que se pueden reproducir con este chip, y los códigos decimales y hexadecimales que debe cargar para reproducir cada uno. Si una vez que haya construido este proyecto quiere hacer algo más fácil de usar, puede construir un circuito más sofisticado que incluye una ROM extra y que lo distribuye GI. Esta ROM extra extiende el número de alófonos disponibles. Vea la hoja de características que reproducimos en el apéndice tres.

'Software' para el sintetizador de palabras

Refiriéndonos una vez más a la figura 3.17, puede observar que los códigos disponibles para los alófonos van del 0 al 63 (decimal), 0 a 3F (hexadecimal). Esto significa que en este diseño no se usan los dos bits superiores de los octetos mandados al sintetizador. Recuérdelo por el momento.

En su forma más simple, un programa para hacer que funcione el sintetizador debe realizar los siguientes pasos:

1) Inicializar el chip PPI por el que se lee el estado del sintetizador.

2) Preparar el registro de inflexión con el código de tono deseado.

3) Esperar hasta que la señal LREQ negativa del chip sintetizador esté a cero.

Dec	Hex	Fonético	Ejemplo	Dec	Hex	Fonético	Ejemplo
00	00	Silencio	10 miliseg	32	20	aw	Bout
01	01	Silencio	30 miliseg	33	21	dd2	Doom
02	02	Silencio	50 miliseg	34	22	gg3	Rig
03	03	Silencio	100 miliseg	35	23	vv	Void
04	04	Silencio	200 miliseg	36	24	gg1	Gone
05	05	Oy	Toy	37	25	sh	Shine
06	06	i	Fry	38	26	zh	Azure
07	07	eh	Send	39	27	rr2	Bronze
08	08	KK3	Coast	40	28	ff	Fail
09	09	pp	Pat	41	29	kk2	Scott
10	0A	jh	Lodge	42	2A	kk1	Cost
11	0B	nn1	Win	43	2B	zz	Zulu
12	0C	ih	Wit	44	2C	ng	Anger
13	0D	tt2	Trill	45	2D	ll	Lazy
14	0E	rr1	Roast	46	2E	ww	West
15	0F	ax	Suck	47	2F	xr	Prepare
16	10	mm	Much	48	30	wh	Where
17	11	tt1	Post	49	31	yy1	Yes
18	12	dh1	They	50	32	ch	Chain
19	13	iy	Tree	51	33	er1	Better
20	14	ey	Rage	52	34	er2	Bird
21	15	dd1	Good	53	35	ow	Show
22	16	uw1	Stew	54	36	dh2	Thought
23	17	ao	Bought	55	37	ss	Roast
24	18	aa	Hot	56	38	nn2	No
25	19	yy2	Yawn	57	39	hh2	How
26	1A	ae	Platt	58	3A	or	Bore
27	1B	hh1	Hurl	59	3B	ar	Arm
28	1C	bb1	Rib	60	3C	yr	Beer
29	1D	th	Thirst	61	3D	gg2	Gate
30	1E	uh	Crook	62	3E	el	Meddle
31	1F	uw2	Brood	63	3F	bb2	Boil

NOTA: Como se puede observar, este chip reproduce alófonos pertenecientes al idioma inglés. Como hasta ahora no se ha fabricado ningún chip que sea aplicable a nuestro idioma, lo más prudente será hacer pruebas para conseguir los sonidos más parecidos a los que empleamos nosotros. Así que puede ser un buen ejercicio que le dará otra satisfacción aparte de la construcción del circuito.

Fig. 3.17 Lista de alófonos del chip SP0256

4) Mandar el código del siguiente alófono que se quiera emitir.

5) Comprobar si el código que acabamos de mandar es el último. Volver al paso 3 si no lo es; si lo es, terminar.

En la figura 3.18 tenemos el diagrama de flujo; en la figura 3.19 listamos el programa en BASIC, que es una implementación del diagrama de flujo. Se incluyen varias sentencias DATA que hacen que el circuito diga: "I am CPC464, but you can call me Arnold" (Yo soy CPC464, pero me puedes llamar Arnold). Observe la monotonía de la frase.

¿Recuerda los bits que no se usaban? supongo que se acuerda, son los que no se usaban en los códigos de alófonos. Ahora podemos usarlos para algo interesante en un programa más sofisticado que interpretará cada dato de la lista de sentencias DATA de una de las cuatro

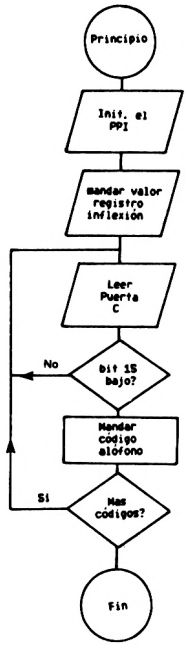


Fig. 3.18 Diagrama de flujo del programa para el sintetizador

```
100  '** Programa de demostración del sintetizador de palabras **
200  OUT &F8F8,0 '  Hace callar a Arnold
300  RESTORE: MODE 2: ON BREAK GOSUB 1400:
     PRINT "Programa de demostración del sintetizador de palabras con
     selección manual de inflexión";STRING$(4,10): INPUT "Seleccione tono
     de inflexión (0 a 7)";SPR
400  IF SPR < 0 OR SPR > 7 THEN 300 ELSE OUT &F8F9,SPR ' Prepara el
     registro de inflexión
500  OUT &F8F3,&AF ' Prepara el PPI para poder ver en PC5 el bit de ocupado
     del chip sintetizador.
600  READ D: IF D > 63 THEN 1300 ELSE WHILE (INP(&F8F2) AND 32) <> 0:
     WEND: OUT &F8F8,D: GOTO 600
700  DATA 4,4,6,4,26,16,3,55,19,3,9,19,3,55,19,3,40,58,4,55,55,
     12,12,41,55,3,40,58,4,4,4
800  DATA 4,4,28,15,17,3,25,53,4,42,26,11,4,42,23,45,3,16,19,4
900  DATA 59,56,23,45,33,4,4,4,4,4,4
1000 DATA 4,4,4,4,6,3,46,24,11,13,3,49,31,3,23,45,3,13,31,3,56,
     53,3,3,4,29,26,17,3,4,6,3,26,16,4,40,19,45,12,11,61,4,35,7,
     47,19,3,33,12,9,14,7,55,55,13,4,4,4,4,4,4,4
1100 DATA 4,4,4,4,4,4,4,53,4,4,46,15,11,4,4,13,31,4,29,14,19,4,
     40,58,4,40,6,35,4,55,55,12,12,41,55,4,55,55,7,39,15,11,4,20,
     2,17,4,56,6,12,4,13,7,11,4,4
1200 DATA 4,4,27,26,45,53,46,4,4,4,4,4,4,4,27,26,45,53,46,4,4,4,4,4,
     4,4,64
```

```
1300  MODE 1: LOCATE 10,10: PRINT "Arnold ha hablado!": T=TIME:
      WHILE TIME < T+400: WEND: CLS: LOCATE 10,10:
      PRINT "Chissst! va a volver a hablar.....": RESTORE: GOTO 600
1400  OUT &F8F8,0 ' Calla a Arnold. La próxima vez termina
1500  MODE 2: PRINT "El poderoso se ha callado....";STIRNG$(4,10):
      END
```

Fig 3.19 Programa BASIC para manejar el sintetizador

formas siguientes:

1) Un código de un alófono

2) Un valor a cargar en el registro de inflexión

3) Un valor que se usa en un bucle de retardo, para permitir que se continúe emitiendo el alófono actual durante la duración del bucle.

4) Un valor que significa que se ha llegado al final de la lista de códigos de alófonos.

Por lo tanto, vamos a usar esos dos bits como instrucciones para el programa que hace el READ de los datos que hay en las sentencias DATA. (NOTA: Si va a extender alguna vez el juego de alófonos usando el circuito con la ROM que mencionamos anteriormente, se usarán todos los bits de los códigos de alófonos). Podemos dibujar una pequeña tabla para reflejar las combinaciones posibles que se pueden formar con estos dos bits.

Bit 7	Bit 8	Significado
0	0	Los 6 bits restantes son un código de alófono
0	1	Fin de la lista de alófonos
1	0	Los 3 bits inferiores son el código para el registro de inflexión
1	1	Prolongar el último alófono. los bits 0-6 se multiplican por 5 y el resultado se usa como contador de retardo. Durante el retardo suena el alófono en curso

```
50    ' ** Programa de demostración del sintetizador de palabras - Versión
        con auto-inflexión
100   RESTORE: MODE 2: ON BREAK GOSUB 1800:
      PRINT "Programa de demostración del sintetizador versión
      auto-inflexión";STRING$(4,10)
200   OUT &F8F8,0 '  Hace callar a Arnold
300   OUT &F8F3,&AF ' Prepara el PPI para ver en PC5 el bit de ocupado del
      chip sintetizador.
400   READ D: CTRL=((D AND &C0)/64): IF CTRL = 0 THEN
      GOTO 900
500   '
      **        Rutina para manejar los códigos, cuando los dos bits    **
      **       superiores NO son cero. Estos se guardan en CTRL          **
600   '**       y tienen el siguiente uso. 00 = ilegal: 01 = final       **
      **       02 = hacer un AND del código con 07 y mandarlo al         **
```

```
650  '**       registro de inflexión:                                **
     **        03 = prolongar el último alófono.                     **
     **        Los bits 0-6 de D*2 = valor del bucle                 **
700  IF CTRL = 1 THEN GOTO 1700 ELSE IF CTRL = 2 THEN
     OUT &F8F9,(D AND 7) ELSE IF CTRL = 3 THEN FOR I=0 TO
     ((D AND 63)*5): NEXT I
800  GOTO 400
900  IF (INP(&F8F2) AND 32) <> 0 THEN 900
1000 OUT &F8F8,D: GOTO 400
1100 DATA 4,4,133,6,4,132,26,16,3,55,19,3,9,19,3,55,19,3,40,
     58,4,55,55,12,12,41,55,3,40,58,4,4,4
1200 DATA 4,4,28,15,&F8,17,3,25,53,&F8,4,42,26,11,&F8,4,42,23,
     &FF,45,&CF,3,16,19,4
1300 DATA 133,59,&FC,56,23,45,33,4,4,4,4,4,4
1400 DATA 4,4,4,4,131,6,3,46,24,11,13,3,49,31,3,23,&F0,45,3,13,31,3,130,
     56,53,255,3,3,4,29,26,17,3,4,6,3,26,16,4,40,19,45,12,11,61,4,129,35,
     7,47,19,3,33,12,128,9,&F0,14,7,55,55,13,4,4,4,4,4,4,4
1500 DATA 4,4,4,4,4,4,4,&82,53,4,4,46,15,&C8,11,&C8,4,4,13,31,4,29,14,
     19,&FF,4,40,58,4,40,6,35,4,55,55,12,12,41,55,4,55,55,7,39,15,11,4,
     20,2,17,4,56,6,12,4,13,7,11,4,4
1600 DATA 4,4,&87,27,26,&EF,&82,45,53,&FF,&F0,46,4,4,4,4,4,4,4,
     &80,27,26,&EF,&85,45,53,&FF,&F0,46,4,4,4,4,4,4,4,64
1700 MODE 1: LOCATE 10,10: PRINT "Arnold ha hablado!": T=TIME:
     WHILE TIME < T+400: WEND: CLS: LOCATE 10,10:
     PRINT "Chissst! va a volver a hablar.....": RESTORE: GOTO 400
1800 OUT &F8F8,0 ' Calla a Arnold. La próxima vez termina
1500 MODE 2: PRINT "El poderoso se ha callado....";STRING$(4,10):
     END
```

Fig. 3.20 Programa BASIC mejorado para manejar el sintetizador

La figura 3.20 es el programa que usa este esquema, incluye los códigos que hacen que el sintetizador hable con un poco mas de 'sentimiento', las primeras sentencias DATA son las mismas que en el programa anterior, pero se han añadido unas cuantas más para hacerle una demostración del resultado del registro de inflexión.

El programa es casi autoexplicativo, pero hay algunos puntos que pueden necesitar aclaración. En la línea 300 se pone el modo del PPI como Hex AF. Esto hace que el PPI se ponga en un modo en el que la línea PC5 de la puerta C se pueda usar como entrada. Como puede ver en las figuras 3.14 y 3.15, la línea LREQ negativa se prepara para poder ser interrogada mediante PC5. La variable CTRL se obtiene a partir de la variable R, que contiene el valor de DATA que se acaba de leer. Cuando se ejecuta la línea 400, la variable CTRL contiene los dos bits altos del contenido de R, pero se han desplazado y ahora son los dos bits bajos del octeto. Esto se hace mediante un AND de los dos bits superiores de R y dividiendo el resultado por 64. Como en:

```
X=&40: X=(X/64): PRINT X
```

Primero se prepara X para que sea = Hex 40 (01000000 en binario), Después se divide X por 64, que hace el mismo efecto que hacer un desplazamiento de todos los bits 6 lugares hacia abajo - introduciendo ceros por la izquierda a medida que tiene lugar el desplazamiento, el bit de más a la derecha desaparece. Si aplicamos esto al número binario 01000000, sucederá lo siguiente:

```
Desplazamiento #1: 00100000
Desplazamiento #2: 00010000
Desplazamiento #3: 00001000
Desplazamiento #4: 00000100
Desplazamiento #5: 00000010
Desplazamiento #6: 00000001 = 1; así que X es ahora = Hex 01
```

Número E	Disposit.	Descripción	Patillas aliment. +5V	+12V	-12V	OV
1	74LS20	2 puertas NAND 4 entradas	14	--	--	7
2	74LS138	Decod. de 3 a 8 líneas	16	--	--	8
3	8255A	Chip PPI	26	--	--	7
4	74LS04	Inversores Hex	14	--	--	7
8	74LS32	4 puertas OR 2 entradas	14	--	--	7
9	SPO-256	Sintetizador de palabras	7	--	--	1
10	74LS174	Retención datos hex.	16	--	--	8
11	74LS629	Chip VCO	16	--	--	8

Fig. 3.21 Tensiones de alimentación de los chips de este proyecto

Cuando la línea 400 ha extraído los dos bits superiores y los ha ajustado a una forma más fácil de usar, de la forma descrita, se comprueba si el resultado es cero. Si lo es, significa que los 6 bits inferiores de la variable R contienen un código de alófono, por lo tanto se hace un GOTO 900. Si el resultado no es cero, las líneas 700 y 800 realizan la acción requerida de acuerdo al esquema que hemos decidido antes. El BREAK se intercepta de forma que el sintetizador se pueda silenciar antes de que termine el programa. En mi ejemplo usamos valores decimales para los alófonos en las sentencias DATA, excepto cuando el dato no es un alófono, en cuyo caso usamos una notación Hex. Con esto conseguimos dos cosas: que los códigos de control sean más fáciles de localizar en la lista, y que sea
sea más fácil de calcular su valos.

Condensadores

C1	50 mfd 16 V electrolítico
C4-C5	0.1 mfd 16V disco cerámico
C6	26 pf 16 V poliestireno
C7	0.1 mfd 16 V de cualquier tipo
C8 y C9	0.22 mfd 16 V

Resistencias

R1-R8	10K Ohm 1/4 watio
R9	470 Ohm 1/4 watio
R10 y R11	10K Ohm 1/4 watio
RV1	Potenciómetro de 47K para circuito impreso

Semiconductores

E1	TTL 74LS20 2 puertas NAND de 4 entradas
E2	TTL 74LS138 decodificador de 3 a 8 líneas
E3	8255A chip PPI

E4	TTL 74LS04 inversores Hex
E8	74LS32 cuatro puertas OR de dos entradas
E9	SPO-256
E19	74LS174 Divisor por 6
E11	74LS629 oscilador controlado por tensión

Varios

Zócalos para todos los chips
Conector de expansión del 'bus'
Circuito impreso

Fig. 3.22 Lista de componentes del sintetizador de palabras

Con esto concluimos la descripción del sintetizador de palabras, la figura 3.21 nos muestra las patillas de alimentación que usa cada chip. La figura 3.22 es la lista de componentes del proyecto.

PROYECTO 3: UN TECLADO EXTERNO

Si observa las páginas de una revista de electrónica, seguro que encontrará por lo menos un anuncio de teclados para ordenador. Esto se debe en parte a que las compañías de ordenadores venden los restos de producción, y en parte a la velocidad con que se diseñan nuevos modelos. Si conoce a alguien que trabaje en una industria de ordenadores, seguro que le podrá conseguir un teclado de desecho.

Pero, ¿para qué queremos un segundo teclado QUERTY? Se nos vienen a la cabeza multitud de aplicaciones. Hay ciertas clases de juegos para ordenadores donde pueden tomar parte varios jugadores, y todos pueden ver la pantalla, pero nadie debe ver lo que el otro introduce por el teclado. Esto puede ser códigos secretos, palabras clave, coordenadas de un juego de batallas, etc.

Otra posibilidad, dado que la pantalla del CPC se puede dividir fácilmente en dos partes, se puede partir la pantalla y usar dos teclados, con algo de programación - escrita en BASIC - para realizar una implementación barata de una máquina para dos usuarios. Por supuesto que tendrá dificultades para algunas aplicaciones, pero para entrada de texto puede ser bastante interesante.

Otro uso para una puerta de conexión de otro teclado puede ser añadir un teclado especial para ser usado por minusválidos, ya sea con rótulos grandes en las teclas, o teclados para usarlos con alguna otra parte del cuerpo que no sean las manos. No he diseñado ninguna unidad de este tipo, pero sería bastante fácil construir un sintetizador de voz dentro de una unidad aislada conteniendo un teclado externo, que se conectaría al CPC, esto sería de gran utilidad para los ciegos, que les diría la tecla que han pulsado.

Estas son algunas de las muchas aplicaciones posibles que me vienen a la mente. No me cabe duda de que a los lectores se les ocurrirán bastantes más.

Estos teclados de los anuncios de las revistas de electrónica, se dividen en dos categorías que se llaman codificados y no codificados. En este contexto, codificado significa que la unidad del teclado tiene suficientes componentes electrónicos como para presentar al ordenador el código de la tecla que se ha pulsado. Un teclado no codifi-

cado - como el del CPC - es analizado por programa. Este segundo método significa que el ordenador debe interrogar continuamente cada tecla del teclado para ver si se ha pulsado, y el proceso se debe ejecutar continuamente mientras está encendido el ordenador. Esto no es tan malo como puede parecer al principio, debido a la diferencia de velocidad entre la persona que teclea y el ordenador.

Hay disponibles muchos chips que realizan la interrogación al teclado como una función de 'hardware'. Estos chips se encuentran normalmente dentro de la unidad de un teclado codificado. Lo que hacen es detectar la tecla que se ha pulsado y mandar al ordenador el código correspondiente de esa tecla - usualmente ASCII. (En el apéndice 4 hay una lista de los códigos ASCII).

Debido a la velocidad de los circuitos electrónicos, algunas veces resulta difícil conectar relés o interruptores a ellos. Aunque usted pueda pensar que un micro-interruptor tiene una acción rápida, una visión a cámara lenta de los contactos nos mostrará que estos rebotan cuando se tocan. Esto significa que se abren y cierran varias veces, antes de que se cierren realmente. En aplicaciones como encendido y apagado de lámparas no es importante, ya que la duración del rebote es de milisegundos. Sin embargo, si pensamos que un chip lógico puede contar hasta 25 millones de impulsos por segundo, veremos que esta situación es crítica cuando el interruptor se conecta a un ordenador. Si no se tuviera en cuenta este efecto, podría pulsar una tecla y obtener el código cuatro o cinco veces. ¿Cómo se podría evitar esto? Se han desarrollado diversas técnicas para evitar estos rebotes. La solución de 'hardware' suele incluir un retardo de tiempo con una red CR en la entrada lógica. Esto hace el efecto de planchar el rebote. La solución por programa es comenzar con un pequeño bucle de retardo cuando se realiza el primer contacto, y esperar el suficiente tiempo para que se detenga el rebote. A esto se le suele llamar un tiempo de asentamiento, o bucle de retardo antirrebote. La situación no es tan mala cuando se usan teclas de pulsadores, ya que están especialmente diseñados para un mínimo rebote de los contactos, aunque con estos pulsadores todavía existe el problema.

Cuando escoja un teclado debe buscar tres cosas.

1) ¿Tiene un diagrama de conexiones? Si no lo tiene no lo compre, ya que va a tener que adivinar las conexiones, y no es muy fácil.

2) ¿Está codificado - tiene un chip codificador dentro? Si lo tiene, ¿es un codificador ASCII? Si no lo es la conexión a su CPC será más difícil - ya que usted tendrá que escribir una rutina de conversión.

3) ¿Que tensiones de alimentación necesita? Todos necesitan +5V y OV, pero algunos necesitan también -12V. Si ha construido nuestra fuente de alimentación, esto no será un problema.

Para resumir, necesita un teclado codificado en ASCII, con detalles de conexionado y de requerimientos de alimentación.

La figura 3.23 es un esquema que se usa para tres de los proyectos de nuestra tarjeta múltiple. Estos son, la puerta del teclado externo (que estamos viendo ahora), la puerta de la matriz de teclas, y el registro de sensibilidad. Los dos últimos se describirán en las dos próximas secciones.

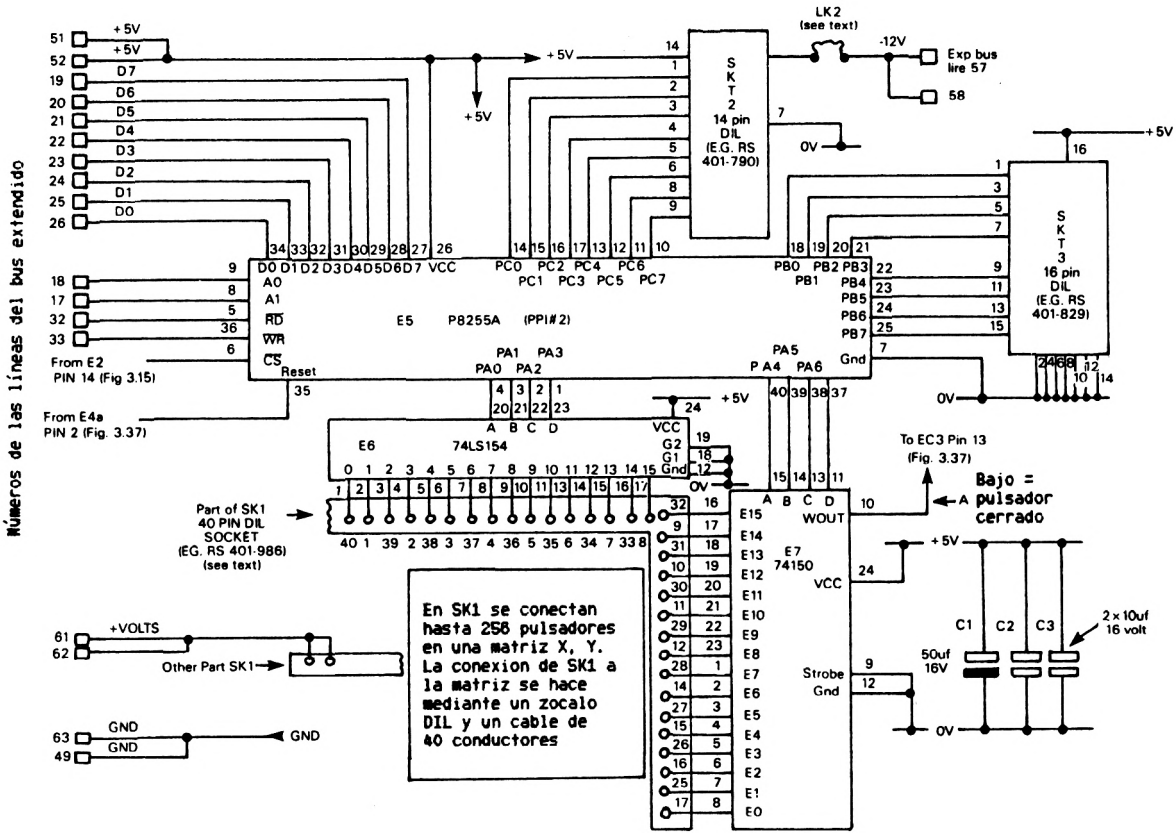


Fig.3.23:Esquema de la segunda parte de la tarjeta multiproyecto

En la figura 3.23, el dispositivo central es el segundo chip 8255 PPI. Este está programado de forma que la puerta A sea una puerta de salida de 8 bits, mientras que las puertas B y C sean puertas de entrada de 8 bits.

La puerta B se usa para el teclado exterior. Ya hemos visto en la figura 3.15, el decodificador de direcciones de la tarjeta multi-proyecto. Refiriéndonos brevemente a ella, se puede ver que la salida Y1 del chip decodificador E2 se pone a cero para seleccionar el segundo PPI. Esto pone los registros del PPI #2 con las siguientes direcciones de E/S:

Registros del PPI #2

Puerta A	Hex F8F4
Puerta B	Hex F8F5
Puerta C	Hex F8F6
Registro CTRL	Hex F8F7

Esto significa que, por ejemplo, los comandos:

```
PRINT INP(&F8F6): OUT &F8F6,10
```

Estos comandos leen y escriben en la puerta C. Esta puerta es a la que vamos a conectar nuestro proyecto de teclado externo.

Cuando vaya a comprar un teclado se encontrará con que el 99% de ellos tienen 7 líneas de datos y una de sincronismo. Los siete bits de datos contienen el código ASCII de la tecla pulsada, y la línea de sincronismo lanza un impulso para indicar que hay datos nuevos disponibles. Si, como en el caso de nuestro diseño, se introducen esas líneas dentro de una puerta sin circuito de retención, el programa debe comprobar regularmente si hay un impulso de sincronismo activo del teclado.

La disposición más usual es usar el bit 7 como señal de sincronismo y los 7 bits restantes como entrada de códigos ASCII. En la figura 3.24 podemos ver el programa BASIC que explora la puerta del teclado. El PPI se prepara para el modo de operación deseado - en este caso el octeto de control es Hex 8B. Después se explora repetidamente el bit 7 de la puerta C para detectar un cambio de nivel lógico. De esta forma podemos detectar señales de sincronismo positivas y negativas - tenga en cuenta que, la mayoría de las líneas de sincronismo de los teclados, está activa cuando está en nivel bajo. Finalmente, cuando se ha detectado una señal activa de sincronismo, se lee en la variable V el valor obtenido de la puerta C, y se pone el bit de sincronismo a cero. El valor así obtenido es el código ASCII del carácter tecleado en el teclado externo. El valor se imprime en la pantalla del CPC - tenga en cuenta que el punto y coma después de la sentencia PRINT suprime el retorno de carro. Pulsando cualquier tecla en el teclado del CPC se para el programa.

Refiriéndonos de nuevo a la figura 3.23: el teclado externo se conecta a la tarjeta multi-proyecto por SKT2, que es un zócalo DIL estándar de circuito integrado. Este es un conector bueno y barato. Se suelen usar cuando las conexiones no se desconectan frecuentemente, pero no son recomendables cuando se deben desconectar periódicamente. RECUERDE poner una marca en los dos conectores para no equivocarse al conectarlos. Debe comprar la suficiente cantidad de cable para el teclado (unos 3 metros máximo) y conectarlo en el conector DIL - vea la

lista de componentes.

Después conecte al teclado los conductores apropiados del cable de cinta. No se proporcionan detalles del método de conexión en el otro extremo del cable, ya que obviamente depende del tipo de conector con el que venga el teclado. Los usos del conector DIL son los siguientes:

SKT2 conexiones por número de patilla

Patilla	*Uso*	*Patilla*	*Uso*
1	Datos bit 0	8	Datos bit 6
2	Datos bit 1	9	Sincronismo del teclado
3	Datos bit 2	10	sin usar
4	Datos bit 3	11	sin usar
5	Datos bit 4	12	-12 voltios al teclado
6	Datos bit 5	13	sin usar
7	0 voltios (Tierra)	14	+5 voltios al teclado

El puente LK2 que se ve en el esquema es para extender, o no, los -12V al teclado. Muchos teclados no requieren mas que los +5 voltios y la masa. Si el que tiene usted es de este tipo, quite el puente LK2. Esto elimina cualquier peligro de daño al teclado, en caso de una conexión incorrecta con los -12 voltios conectados.

```
10    '** Programa BASIC para representar en pantalla los caracteres    **
      ** tecleados en el teclado externo
20    OUT &F8F7,&9B ' ** Inicializa el PPI
30    STROBE =128 '** este es para los impulsos de sincronismo positivos,
      para los negativos ponga STROBE a cero
40    WHILE (INP(&F8F6) AND 128) <> STROBE: WEND '** Esperar hasta que
      aparezca el sincronismo
50    PRINT CHR$(INP(&F8F6) AND 127); : GOTO 40
32767 END
```

Fig. 3.24 Programa para probar el teclado externo

Detalles de construcción

Maneje los chips con cuidado. Como ya hemos dicho, el método exacto de hacer las diez u once conexiones desde la puerta al teclado, dependen enteramente del tipo de teclado que tenga usted. Si consigue los datos del teclado cuando lo compre, no tendrá dificultad en la interconexión con la puerta. La figura 3.25 muestra las patillas de alimentación de los dispositivos usados. La figura 3.26 es la lista de componentes del proyecto.

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	0V
1	74LS20	2 puertas NAND 4 entradas	14	--	--	7
2	74LS138	Decod. de 3 a 8 líneas	16	--	--	8
4	74LS04	Inversores Hex	14	--	--	7
5	8255A	Chip PPI	26	--	--	7

Fig. 3.25 Tensiones de alimentación de este proyecto

Esta es la lista de componentes que se requieren para construir la sección de la puerta para teclado externo, en la tarjeta multiproyecto.

Condensadores

C1	50 mfd 16 V electrolítico
C2-C3	0.1 mfd 16V disco cerámico

Semiconductores

E1	TTL 74LS20 2 puertas NAND de 4 entradas
E2	TTL 74LS138 decodificador de 3 a 8 líneas
E4	TTL 74LS04 inversores Hex
E5	8255A chip PPI

Varios

Zócalos para todos los chips

SK2	zócalo DIL de 14 patillas

Conector de expansión del 'bus'
Circuito impreso

Fig. 3.26 Lista de componentes para la puerta de teclado externo

PROYECTO 4: PUERTA PARA MATRIZ DE TECLADO

Este proyecto permite la conexión de gran cantidad de interruptores sin necesidad de usar mucha electrónica para la interconexión. Este proyecto encontrará muchas aplicaciones en sistemas computerizados que requieren la exploración de gran cantidad de interruptores. Como ejemplo tenemos los teclados adaptados para el cliente, teclados sin codificar, alarmas antirobo - en donde los interruptores se fijan a las ventanas y puertas, y también detectores conectados al ordenador. Otra aplicación que me parece interesante, pero para la que no he tenido nunca tiempo, dinero u oportunidad de intentarla, es comprar una máquina vieja de bolas, quitarle las partes electromecánicas y reconstruirla usando un sistema electrónico cuyo elemento central sea un microprocesador. En este tipo de proyecto exploraríamos los interruptores de detección de la bola con un circuito como este. Otra aplicación serían los juegos, y se incluye el listado de un pequeño programa de "Campo Minado" para usarlo con este proyecto. Otra aplicación que me entusiasmaría sería la exploración de varios detectores para un tren eléctrico. Es suficiente decir que si usted necesita explorar repetidamente un gran número de interruptores de dos contactos, lo puede hacer con este circuito.

Como reside en la tarjeta multi-proyecto, la Puerta de Matriz de Teclas (KMP) usa el decodificador de direcciones de la misma forma que la puerta del teclado en el último proyecto. Cuando usted accede a la dirección de E/S Hex F8F4 está escribiendo o leyendo en la puerta A del PPI #2. El KMP vive en la puerta A, que se define como puerta de 8 bits de salida. Refiriéndonos de nuevo a la figura 3.23 vemos que la puerta A tiene dos chips conectados. Estos son E6 y E7. El chip E6 es un decodificador de 4 a 16 líneas, para usarlo debe introducir su código binario por sus entradas A, B, C y D, a la salida tendremos a cero la patilla correspondiente al número que coincida con el binario de entrada. El chip E7 es un selector de datos de 16 canales. Tiene 16 entradas - E0 a E15, y una salida llamada WOUT. Este chip tiene también entradas A, B, C y D, pero en este caso seleccion cuál de las entradas de E se mandarán a la patilla WOUT.

Las entradas ABCD al chip E6 se introducen con códigos binarios desde los cuatro bits inferiores de la puerta A, mientras que las entradas ABCD de E7 se introducen desde los cuatro bits superiores de la misma puerta. Esto significa que si iniciamos un bucle FOR NEXT desde 0 a 255 y como parte de cada bucle sacamos el valor del bucle a la puerta A, hacemos que se pongan a cero por turno cada una de las salidas de E6, y cada vez que los cuatro bits inferiores se pongan a cero habremos añadido uno al contador de los cuatro bits superiores, cambiando así la entrada de E7 que va a la patilla WOUT de E7. (Observe que WOUT siempre contiene el nivel lógico inverso de la entrada seleccionada). Si conectamos, por medio del zócalo de extensión SK1, el número de interruptores necesarios con un terminal conectado a una salida de E6, y el otro terminal conectado a una entrada de E6, y pulsamos después un interruptor, seremos capaces de detectar la pulsación parando la exploración cuando veamos que la patilla WOUT se ha puesto a cero. ¿Porqué debemos fijarnos en WOUT? Esta patilla está conectada al bit 4 de la puerta C en PPI #1 y está en la dirección de E/S F8F2. Por ejemplo, si queremos leer el estado del interruptor que tiene un terminal conectado a Y fila 8 (Patilla 9 de E6), y el otro conectado a X fila 3 (patilla 5 de E7), usaríamos el siguiente comando para inicializar los PPIs y leer el estado de los interruptores direccionados:

```
OUT &F8F3,&AE: OUT &F8F7,&8B: OUT &F8F4,&83: STATE=INP(&F8F2):
IF (STATE AND &10)=0 THEN PRINT "No pulsado" ELSE PRINT "Pulsado"
```

La patilla 40 del zócalo DIL SK1 se usa para la conexión de una matriz externa de teclas, o un punto de distribución, si los interruptores no son locales. Una cosa importante a tener en cuenta es que para ciertas aplicaciones puede ser una buena idea conectar algunas resistencias - de unos 10K7 ohmios desde las líneas X a los +5 voltios. Esto se hace para asegurars que los cables largos de los interruptores remotos no producirán problemas de ruidos. Esta es la razón para extender los +5 voltios a la distribución. La conexión de las patillas de SK1 es como sigue:

Patilla 1	Y1	Patilla 21	+5 voltios
Patilla 2	Y3	Patilla 22	--
Patilla 3	Y5	Patilla 23	--
Patilla 4	Y7	Patilla 24	--
Patilla 5	Y9	Patilla 25	X1
Patilla 6	Y11	Patilla 26	X3
Patilla 7	Y13	Patilla 27	X5
Patilla 8	Y15	Patilla 28	X7
Patilla 9	Y14	Patilla 29	X9
Patilla 10	Y12	Patilla 30	X11
Patilla 11	Y10	Patilla 31	X13
Patilla 12	X8	Patilla 32	X15
Patilla 13	--	Patilla 33	Y14
Patilla 14	X6	Patilla 34	Y12
Patilla 15	X4	Patilla 35	Y10
Patilla 16	X2	Patilla 36	Y8
Patilla 17	X0	Patilla 37	Y6
Patilla 18	--	Patilla 38	Y4
Patilla 19	--	Patilla 39	Y2
Patilla 20	+5 V	Patilla 40	Y0

En este proyecto, y en todos los diseños de la tarjeta multi-proyecto, usamos por lo menos uno de los chips PPI (esto se hace porque es un buen momento para comprobar la interconexión con la CPU).

La figura 3.27 es el programa que comprueba el chip PPI de una forma limitada. Se asegura de que el chip PPI es capaz de tomar y retener cualquier valor, cuando todas sus puertas se ponen como de salida. El programa imprime los detalles de cualquier fallo. Esto nos permite eliminar bits espúreos, o cualquier problema que pudiera ocurrir debido a las malas soldaduras, conexiones, etc. Por favor, lea las REM del programa, ya que varias de ellas son notas muy importantes.

La figura 3.28 es el programa que usará para comprobar, o familiarizarse, con el KMP. Dibuja en la pantalla las 256 teclas posibles en forma de cuadrados. Cuando se pulsa una tecla, el cuadrado correspondiente se rellena de color, y el cuadrado de la tecla pulsada anteriormente parpadea ligeramente. El programa emite también un sonido agudo durante el tiempo que se mantiene pulsada la tecla, y un sonido grave durante el tiempo que está explorando las teclas y no encuentra ninguna pulsada. Observará que el programa, como está escrito en BASIC, tarda algunas veces en responder. Esto se puede rectificar usando una sección de BASIC para la parte de manejo de la pantalla y una subrutina en código máquina para la función de exploración. Esta

solución se ha adoptado para el juego Campo Minado que veremos enseguida.

```
10    '    Programa de comprobación del PPI. Escrito para comprobar
           un PPI cuyo primer registro de direccione se especifica
           en la variable PPIBASE, en la línea 30.
11    '    Comprueba que puede sacarse cualquier valor a cualquiera de
           las puertas PPI.
           ** NOTA: El programa pone todas las puertas como de salida **
12    ' VERSION 2
20    MODE 2          'PONE MI MODO FAVORITO DE PANTALLA
30    PPIBASE=&F8F0   'AQUI PONE LA DIRECCION PPIBASE
40                    'EL PROGRAMA TOMA AHORA PPIBASE COMO DIRECCION DE
                       LA PUERTA A, Y PPIBASE+1 COMO PUERTA B, PPIBASE+2
                       COMO DIRECCION DE PUERTA C, Y PPIBASE+3 COMO
41                    'DIRECCION DE REGISTRO DE CONTROL

45    OUT PPIBASE+3,&80: PRINT "Todas las puertas se ponen como salida":
      PRINT
50    PRINT "QUE COMPROBACION? DESEA? (G)ENERAL O (I)NDIVIDUAL ";
60    T$=INKEY$: IF T$="" THEN 60 ELSE PRINT : IF UPPER$(T$)="G" THEN 100
70    INPUT "A QUE PUERTA MANDAMOS LOS VALORES? ",P$:P$=UPPER$(P$):
      IF P$="A" THEN OA=PPIBASE ELSE IF P$="B" THEN OA=PPIBASE+1 ELSE
      IF P$="C" THEN OA=PPIBASE+2 ELSE PRINT: PRINT "Debe ser A, B o C" :
      GOTO 70
80    INPUT "Valor decimal a poner en la puerta ";VALUE: IF VALUE > 32767
      THEN 80 ELSE OUT OA,(VALUE AND 255): PRINT: PRINT "PUERTA:","A","B",
      "C":PRINT: PRINT "CONTENIDO:";,
90    FOR I=PPIBASE TO (PPIBASE+2): X=INP(I): PRINT CHR$(8);X;
      "(HEX=";HEX$(X);")",;:NEXT I:PRINT STRING$(50,95):PRINT: GOTO 70
100   '  Esta sección comprueba las puertas del PPI seleccionadas para
         salida. Esto lo hace escribiendo un contador en ellas. Así se
         comprueba la lógica, pero hará falta un osciloscopio o un aparato
110   '  de medida para asegurarse de que aparecen realmente los cambios
         en las salidas. No aparecerán en caso de que esté defectuoso un
         transistor de salida (dentro del PPI).
120   CLS: LOCATE 25,1: PRINT "Comienza la prueba general"
130   LOCATE 8,5: PRINT "Valor de prueba =";: FOR I=0 TO 255: FOR X=0 TO 2:
      OUT PPIBASE+X,I: RD=INP(PPIBASE+X): IF RD <> I THEN GOSUB 2000
      ELSE LOCATE 20,5: PRINT I;CHR$(13)
135   NEXT X: NEXT I: PASS=PASS+1: LOCATE 1,3: SOUND 1,30,10:
      SOUND 2,60,10,14: SOUND 4,120,11,15: PRINT "Pasada ";PASS;
      " completa";CHR$(20);CHR$(13);: GOTO 130
1000  '
                        Rutina de error de impresión

1100  SOUND 1,200,50,15: FAIL=FAIL+1: LOCATE 1,10: PRINT STRING$(80,&2A):
      PRINT TAB(20);"Reporte de error para fallo número ";FAIL;".":
1150  LOCATE 1,13: PRINT "Puerta ";CHR$(65+X);":dirección E/S ";
      HEX$(PPIBASE+X;" bajo prueba."
1200  PRINT "Detalles: Escrito ";I;"(HEX) ";HEX$(I):
      PRINT "Valor leido ";RD;"(HEX) ";HEX$(RD):
      PRINT "Pulse cualquier tecla para continuar"
1300  IF INKEY$="" THEN 2300 ELSE LOCATE 1,6: PRINT CHR$(20): RETURN
35535 END
```

Fig 3.27 Programa de prueba del PPI

```
500  '** Este es un programa para comprobar y demostrar la Puerta de
     ** Matriz de teclas, lea el texto del libro si quiere más detallas
1000 OUT &F8F3,&99: OUT &F8F7,&80 ' INICIALIZA LOS PPIs
1100 ON BREAK GOSUB 2800
1200 INK 2,24,1: SPEED INK 10,25: COL=1: MODE 1
1300 FOR O=64 TO 304 STEP 16: FOR I=64 TO 304 STEP 16
1400 MOVE I,O: DRAWR 16,0,COL: DRAWR 0,16,COL: DRAWR -16,0,COL:
     DRAWR 0,-16,COL: NEXT I: NEXT O ' Dibuja un recuadro con la
     esquina inferior izquierda en la coordenada I,O
1500 L=10: U=18: COL=2: GOSUB 2200: COL=1: LOCATE 25,11:
     PRINT "=Ultima pulsada"
1600 LOCATE 5,2: PRINT "M . S . D (HEX)": LOCATE 5,5: PRINT STRING$(16,
     245): FOR I=0 TO 15: LOCATE I+5,4: PRINT HEX$(I): NEXT: FOR I=15 TO
     0 STEP -1: LOCATE 3,6+(&F XOR I): PRINT HEX(I);CHR$(152): NEXT
1700 LOCATE 1,8: PRINT "L": PRINT: PRINT ".": PRINT: PRINT "S": PRINT:
     PRINT ".": PRINT: PRINT "D": FIRST=255
1800 GOSUB 2500 ' Ir a la subrutina de exploración de teclas, que no
     devolverá control hasta que se haya pulsado una tecla. El número de
     la tecla aparecerá en la variable I cuando retorne.
1900 FOR XX=1 TO 3: SOUND 1,50,5,3: SOUND 1,20,5,3: NEXT XX: IF FIRST
     <> 0 THEN 2000 ESLE IF I=LASTKEY THEN GOTO 1800 ELSE COL=1
1910 GOSUB 2200 ' Sonido que indica que se ha pulsado una tecla. Ver si
     es la primera pasada, si lo es
2000 U=(I AND &F0)/16: L=I AND &F
2100 FIRST=0: COL=2: LASTKEY=I: GOSUB 2200: GOTO 1800 ' Restaurar el
     primer señalizador, poner la tinta parpadeante, recordar esta
     tecla, parpadear la caja y volver a explorar
2200 '
     ** Esta es la subrutina que rellena un recuadro. La tinta usada se
        indica en la variable COL. Si es = 2, parpadeará **
2300 MOVE 64+(U*16),64+(L*16): FOR XX=0 TO 16: DRAWR 16,0,COL: MOVER 0,1:
     DRAWR -16,0,COL: NEXT XX
2400 COL=1: RETURN
2500 '
     repetir hasta que el bit 4 de la puerta C de PPI#1 (Ver esquema) sea
     un 1 lógico. Esto significa que se ha pulsado la tecla número I

2600 FOR I=0 TO 255: OUT &F8F4,I: IF (INP(&F8F2) AND (&10)) <> 0
     THEN RETURN
2700 NEXT I: SOUND 1,200,20,3: GOTO 2600 ' La sentencia SOUND indica que
     se ha completado una exploración sin novedad. Bórrelo si le molesta.
2800 MODE 2: END
```

Fig 3.28 Programa de familiarización con la puerta de matriz de teclas

Un punto final acerca de la parte 'hardware' del KMP. Aunque el diseño permite la conexión de 256 interruptores en la matriz, solo debe añadir los que necesite. Cuando conecte menos interruptores del total permitido, conéctelos a las entradas de E7 con los números más bajos. Tomando del ejemplo anterior con 16 interruptores instalados, conecte un contacto de cada interruptor a la patilla 17 de SK1 (XO), y conecte el otro contacto a la línea Y apropiada, de acuerdo con la tabla de conexiones SKT1. Cuando use menos de 256 interruptores, no olvide modificar el programa en los puntos indicados por las REMs, esto le evitará comprobar interruptores inexistentes, y hará que la parte de exploración del programa se ejecute más rápidamente.

Fig 3.29 Conexión de la matriz al KMP

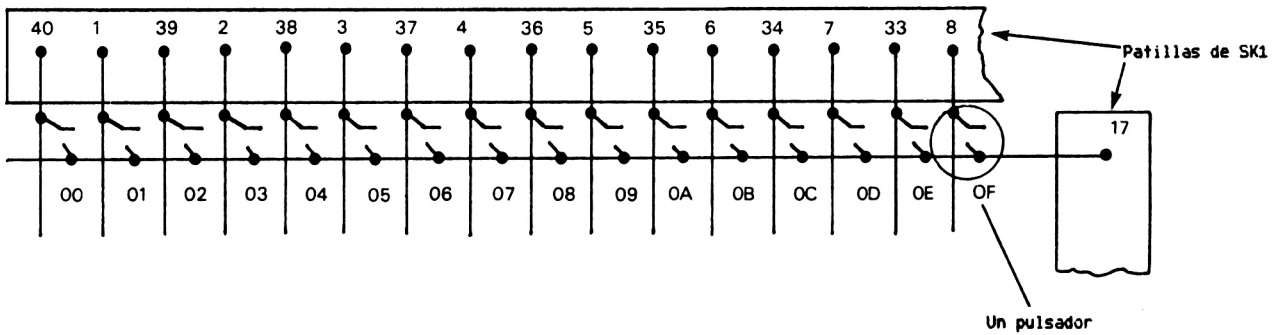
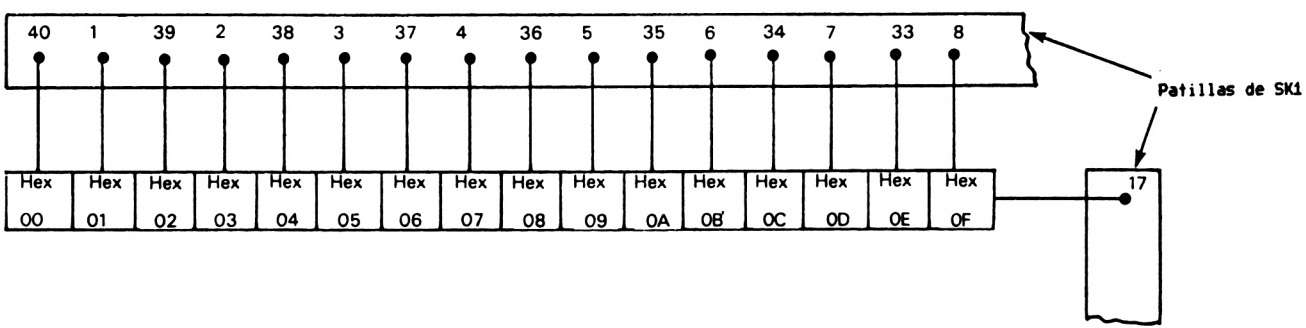


Fig 3.30 Conexión de la matriz al KMP



Ahora pasaremos al juego del Campo Minado. Debe haber comprado, o construido una pequeña matriz de teclas en una caja adecuada, con un mínimo de 16 pulsadores en ella. Los pulsadores se conectan al KMP por medio de un cable de cinta, y un conector DIL. Las conexiones de SK1 se listan en la figura 3.29. Puede construir un teclado mayor que el que se muestra aquí, pero tendrá que modificar el programa en varios los puntos marcados por los REM. Para añadir más pulsadores, conéctelos de la forma requerida en X1-X15. Si sigue la figura 3.29 cuando conecte la matriz de teclas, corresponderá a la figura 3.30, que nos muestra los valores que se deben escribir en la puerta A de PPI #2 para leer el estado de cada uno de los interruptores de la matriz.

El escenario está preparado de forma que dos jugadores deben entrar dentro de un campo de minas cerrado por un malvado (¿lo conoce?). Los jugadores no pueden escapar, pero deben caminar alrededor del campo de minas hasta que uno de ellos desaparezca por el efecto de una explosión. Una vez que ha ocurrido esto, el malvado dejará libre al superviviente. El objetivo es que cada jugador recorra el campo de minas sin pisarlas. El campo es un grupo de recuadros en la pantalla, cada uno representa un pulsador de la matriz. El programa planta minas aleatoriamente, el número plantado es una quinta parte del número total de recuadros especificados en la línea 1300 del programa. Los recuadros se numeran desde el 1 en la esquina inferior izquierda. Cada jugador puede saltar a cualquier recuadro de la matriz, con el pulsador correspondiente. Hay una breve pausa, (para permitirle que se despierte), y después el recuadro explota o se rellena con el color del jugador para indicar que no había una mina. Cada movimiento correcto aumenta cinco puntos en la cuenta del jugador correspondiente. Para que se pueda jugar tanto en el monitor en color como en el monocromo, la posición del jugador número uno se marca con un cuadrado parpadeante, y la del jugador número dos con un cuadrado sin parpadeo. Este es el juego, y el listado se muestra en la figura 3.31.

```
500  '**              C 1985 Kernow Computers                **
     **                   ¡CAMPO MINADO!                    **

1000 OUT &F8F3,899: OUT &F8F7,880: MEMORY 28999: GOSUB 29000
1100 ON BREAK GOSUB 32767
1200 INK 2,20,1: SPEED INK 50,5: COL=1: MODE 1: INK 3,12,3:
     INK 4,12,24
1300 SW=16 '** Esta variable le dice al programa cuantos pulsadores
           ** hay en la matriz. Puede tener hasta 256. Los números
           ** inferiores a 8 causan efectos muy divertidos
1400 IF SW > 256 THEN 32767 ELSE IF SW > 190 THEN SIZ=16 ELSE IF
     SW > 100 THEN SIZ=20 ELSE IF SW > 48 THEN SIZ=24 ELSE SIZ=32
1450 '** Limitamos los pulsadores a 256, después ponemos en SW el tamaño
     ** en recuadros
1410 DIM MINES(256): IF SW > 32 THEN YOFF=32 ELSE YOFF=64
1500 XOFF=YOFF: FOR I=0 TO SW-1: GOSUB 27000: GOSUB 26000: NEXT
1510 '** Dibuja los recuadros - uno por cada interruptor instalado

1600 RANDOMIZE TIME: P(1)=INT(RND*(SW+1)-1): P(2)=INT(RND*(SW+1)-1):
     IF P(1)=P(2) OR P(1)<1 OR P(2)<1 THEN 1600
     '** Selecciona posición aleatoria de comienzo de lso jugadores
1700 FOR LOOP=1 TO 2: I=(P(LOOP)): GOSUB 27000: COL=LOOP: GOSUB 25000:
     NEXT: PEN #0,2: LOCATE 1,3: PRINT CHR$(143);: PEN #0,1:
     PRINT "=Jugador 1:": PRINT: PRINT CHR$(143);"=Jugador 2:"
```

```
1800  FOR LOOP=1 TO (INT(SW/5))
1900  RANDOMIZE TIME: XX=INT(RND*(SW+1)): IF MINES(XX) <> 0 THEN 1900
      ELSE MINES(XX)=255: NEXT LOOP: FIRST=255
2000  CURPLYR=INT(RND*3): IF CURPLYR=0 THEN 2000 ELSE LOCATE 10,24:
      PRINT "Jugador";CURPLYR;"EMPIEZA";: '** Elije el jugador que
      comienza
2100  GOSUB 28000: LOCATE 10,24: PRINT CHR$(20);:
      '** Ir a subrutina exploración teclado. K=tecla pulsada
2200  IF FIRST <> 0 THEN 2300 ELSE IF K=LASTMOV THEN LOCATE 15,24:
      PRINT "Movimiento rechazado": GOTO 2100 '** Ver si es la primera vez
      ** si es asi, puede pulsar interruptor cero. Si el movimiento es
2250  '** el mismo que el último, lo rechazamos brutalmente.
2300  FIRST=0: I=P(CURPLYR): GOSUB 27000: COL=0: GOSUB 25000:
      IF MINES(K) <> 0 THEN 3000 '** Limpia FIRST. Borra la última posición
      Comprueba si hay una mina en la nueva posición.
2400  I=K: COL=CURPLYR: GOSUB 27000: GOSUB 25000: P(CURPLYR)=K:
      IF CURPLYR=1 THEN LOCATE 15,3 ELSE LOCATE 15,5:
      '** Dibujar el jugador en la panta_la, y posicionar el cursor
2500  SCOR(CURPLYR)=SCOR(CURPLYR)+5:
      PRINT "Puntos="; SCOR(CURPLYR);CHR$(18):
      IF CURPLYR=1 THEN CURPLYR=2 ELSE CURPLYR=1
2600  LASTMOV=K: LOCATE 10,24: PRINT "Jugador";CURPLYR;"mueve":
      GOTO 2100
3000  I=K: GOSUB 27000: MOVE (XOFF+(X*SIZ))+2,(YOFF+(Y*SIZ))+2:
      SPEED INK 2,2: FOR I=1 TO 100 STEP 2: DRAWR I+(INT(RND*15)),0,4:
      DRAWR 0,I,3: DRAWR -(I+INT(RND*15)),0,4: DRAWR 0,-(I+(INT(RND*3))),3
3010  SOUND 1,1,1,15,1,1,15: SOUND 2,2000,1,12,1,1,14:
      SOUND 2,1,1,15,1,1,15: NEXT: FOR I=15 TO 1 STEP -1: INK 2,I:
      SOUND 1,1,30,I,1,1,15: INK 3,I: NEXT: SPEED INK 1,255
3100  LOCATE 1,3: PRINT CHR$(18);STRING$(2,10);CHR$(13);CHR$(18):
      LOCATE 1,24: PRINT CHR$(18);:LOCATE 1,3: IF SURPLYR=1 THEN WIN=2
      ELSE WIN=1
3105  PRINT "Jugador";WIN;"ganó: La puntuación era";SCOR(WIN)
3110  T=TIME: WHILE (T+1500) > TIME: WEND: CLS: CLEAR: RUN
3120
      '** El programa ha terminado, las siguientes son subrutinas **

25000 '** Esta subrutina rellena el recuadro con las coordenadas x,y
       ** que recibe en las variables x e y. Añade los desplazamientos de
       ** pantalla XOFF e YOFF para obtener su posición absoluta. Ya que
25010 '** quiere colorear el recuadro sin destruirlo, hay unos cuantos -1 y
       ** +1  en los valores de las expresiones de los DRAWs.
       ** Finalmente la variable COL contiene el color de la tinta que
25020 '** va a usar cuando dibuje. La rutina pone COL a 1 antes de
       ** retornar, esto evita confusiones. COL=2 hace un relleno parpa-
       ** deante.
25030 MOVE (XOFF+(X*SIZ))+2,(YOFF+(Y*SIZ))+2: FOR XX=1 TO SIZ-3:
      DRAWR SIZ-3,0,COL: MOVER 0,1: DRAWR (-SIZ)+3,0,COL: NEXT XX
25040 COL=1: RETURN
26000 '** Esta subrutina dibuja un recuadro, cuyo ángulo inferior izquierdo
       ** está situado en las coordenadas gráficas de la pantalla, pasadas
       ** a la subrutina en las variables x e y. La rutina añade XOFF e
26010 '** YOFF a estas, para trasladarse dentro de las posiciones de la
       ** pantalla gráfica real. El color de pluma usado se pasa en COL.
       ** Las dimensiones reales de cada recuadro (en puntos) se pasa en
26015 '** SIZ, de forma que si SIZ es igual a 32, se dibujará un recuadro
       ** de 32 puntos.
26020 MOVE (X*SIZ)+XOFF,(Y*SIZ)+YOFF: DRAWR SIZ,0,COL: DRAWR 0,SIZ,COL:
      DRAWR -SIZ,0,COL: DRAWR 0,-SIZ,COL: RETURN
```

```
27000 '** Esta pequeña subrutina convierte el número de pulsador en las
      ** coordenadas X e Y para localizar el recuadro de pantalla que
      ** representa dicho interruptor. Asume que los pulsadores se han
27010 '** puesto contínuos. El número de interruptor a convertir se pasa en
      ** la variable I.
27020 Y=INT(I/16): X=I-(Y*16): RETURN
28000 '** Esta subrutina llama a la rutina en código máquina KMPSCAN que
      ** explora todas las posiciones posibles de los interruptores del
      ** KMP y devuelve el código del primero que encuentra pulsado. Si la
28005 '** posición 30001 de memoria es cero, es que no se ha activado ningún
      ** pulsador, pero si no es cero, la dirección 30000 contiene el
      ** número del pulsador activado
28010 CALL 30002: R$=INKEY$: IF PEEK(30001)=0 AND R$="" THEN 28010
     ELSE IF R$=CHR$(5) THEN GOTO 32767 ELSE
     K=PEEK(30000): RETURN ' Si teclea CTRL/E, termina el juego
29000 '** Esta rutina introduce la subrutina en código máquina KMPSCAN

      ** en memoria, a partir de la dirección 30000. Las llamadas a KEYSCAN
      ** se realizan en la línea 28000. (El listado de KEYSCAN lo
29005 ' ** encontrará en el capítulo 5)
29010 FOR I=30000 TO 30055: READ D: POKE I,D: NEXT
29020 RETURN
29050 DATA 224 , 255 , 213 , 245 , 197 , 1 , 244 , 248 , 22 , 0
29060 DATA 237 , 81 , 14 , 242 , 237 , 120 , 230 , 16 , 32 , 13
29070 DATA 14 , 244 , 20 , 32 , 241 , 50 , 48 , 117 , 50 , 49
29080 DATA 117 , 24 , 18 , 237 , 120 , 230 , 16 , 32 , 250 , 61
29090 DATA 32 , 253 , 122 , 50 , 48 , 117 , 62 , 255 , 50 , 49
29100 DATA 117 , 193 , 241 , 209 , 201 , 0 , 0 , 0 , 0 , 0
32767 MODE 2: END
```

Fig 3.31 Listado del juego Campo Minado

Conclusión

El KMP tiene una gran cantidad de posibilidades que no se han explorado aquí. Por ejemplo, si usted construye los elementos principales del diseño en una tarjeta prototipo, cambiando el 74151 por un 74LS159 y pone varios 7404 o transistores en las salidas del 74LS154, puede cambiar la idea completa y acoplar 256 LEDs individuales en lugar de la matriz de pulsadores. Con esto tendrá una gran pantalla de 16 líneas por 16 columnas de puntos, que puede usar como pantalla especial para luces de discoteca, o como una pantalla de estado, capaz de ser leida a distancia. En ambos casos se necesita escribir algún programa para multiplexar.

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	OV
1	74LS20	2 puertas NAND 4 entradas	14	--	--	7
2	74LS138	Decod. de 3 a 8 líneas	16	--	--	8
3	8255A	Chip PPI	26	--	--	7
4	74LS04	Inversores Hex	14	--	--	7
5	8255A	Chip PPI	26	--	--	7
6	7415A	Decodificador 4/16 líneas	24	--	--	12
7	74150	Selector de 16 líneas	24	--	--	12

Fig. 3.32 Tensiones de alimentación de este proyecto

Detalles de construcción

Todos los chips que se usan en este proyecto son TTL o compatibles, lo que significa que resultará bastante difícil dañarlos con electricidad estática. El problema mayor será el cableado de la matriz de interruptores dentro de la caja que haya elegido. Espero que las instrucciones dadas sean suficiente información para que deje de ser un problema. Cuando haya terminado su construcción usa el programa de la figura 3.28 para asegurarse de que lo ha hecho bien. Finalmente, en la figura 3.32, le damos los números de las patillas de conexión de alimentación, y en la figura 3.33 tiene la lista de componentes usados en el proyecto.

Esta es la lista de los componentes que necesita para construir la parte correspondiente a la puerta para la matriz de interruptores.

Condensadores

C1	50 mfd 16 voltios electrolítico
C2-C3	0.1 mfd 16 voltios cerámico de disco

Semiconductores

E1	74LS20 4 puertas NAND de dos entradas
E2	74LS138 decodificador de 3 a 8 líneas
E3	8255A PPI
E4	74LS04 inversores Hex
E5	8255A PPI
E6	74154 decodificador de 4 a 16 líneas
E7	74150 selector de 16 entradas de datos

Varios

Zócalos para todos los chips
SK1 zócalo DIL de 40 patillas
Conector para el bus de expansión
Circuito impreso

Fig. 3.33 Lista de componentes del proyecto KMP

PROYECTO 5: REGISTRO DE SENSORES

Este es un proyecto bastante sencillo, que usa una vez más el PPI para proporcionar entradas al ordenador. Esta es una puerta de entrada de 8 bits de uso general. La razón para llamarla registro de sensores es que vamos a ver la forma de usarlo para analizar el estado de algunos interruptores.

El registro de sensores se construye dentro de la tarjeta multi proyecto,y usa la puerta B del PPI #2. Esta puerta se puede leer con el comando:

```
OUT &F8F3,&8B: PRINT INP(&F8F2)
```

El comando OUT pone el modo de operación dentro del registro de control del PPI. La figura 3.34 es una sección de la figura 3.23, que corresponde al registro de sensores. La puerta B se extiende a un zócalo DIL de 16 patillas - SKT3. En este zócalo se puede conectar un cable de cinta terminado en un conector tipo DIL para conectarlo al zócalo. El otro extremo del cable se conecta a las señales que se quieren analizar. NUNCA conecte a esta puerta una señal que pueda tener una tensión superior a los +5V, o menor de OV. Si lo hace destruirá el chip PPI, y probablemente dañará también su CPC. Puede conectar las señales directamente desde otros circuitos lógicos, pero algunas señales deberán ser convertidas previamente a niveles lógicos por medio de circuitos externos. Veamos ahora algunas aplicaciones de este circuito.

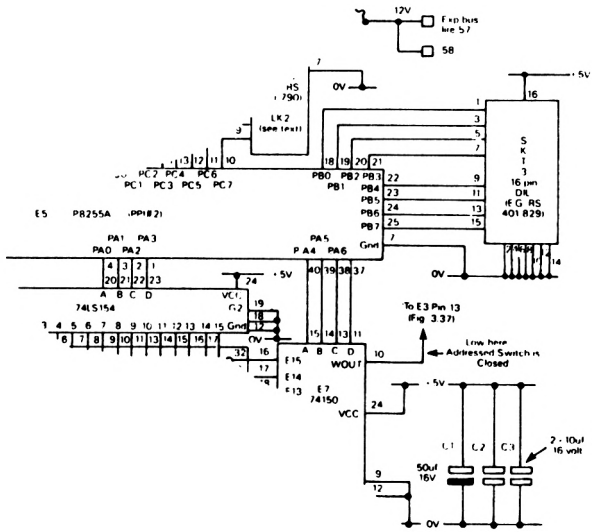


Fig 3.34 Sección de la figura 3.23 con el registro de sensores

La aplicación más simple de este circuito es quizás la de conectarle algunos interruptores tipo DIL. Este tipo de interruptores se suelen

usar en casi todos los periféricos de ordenador para permitir conectar o desconectar ciertos dispositivos. Cuando se enciende el microprocesador que trabaja con la unidad, lee el estado de estos interruptores y actúa de acuerdo con su estado.

Otra aplicación es la de contar impulsos desde los interruptores conectados a la puerta. Esto puede serle útil para una aplicación en la que se tengan que registrar datos, en la que se usa un grupo de ocho pulsadores. Cuando se pulsa alguno, hace que se ejecute un programa en el ordenador, que incrementa un contador asociado a dicho pulsador. Este tipo de aplicación es útil para obtener información de tipo estadístico. Por ejemplo, un observador de aves puede hacerse una pequeña caja de pulsadores para contar cuantos gorriones, lechuzas, abadejos, etc. ha visto en su jardín durante la jornada. Si tiene el ordenador disponible, el costo de la caja de pulsadores para este tipo de trabajo, es bastante asequible. Por supuesto, asumimos que el ordenador se puede conectar a una toma de corriente cerca del lugar de observación.

Echemos una ojeada al programa que nos permitirá contar las veces que hemos pulsado cada uno de los ocho pulsadores conectados al registro de sensores. Primero necesitamos una caja pequeña que nos quepa en la mano, y suficiente cable como para conectarla desde el 'bus' de expansión del ordenador. Los pulsadores de la caja se deben conectar a SKT3, según se muestra en la figura 3.35.

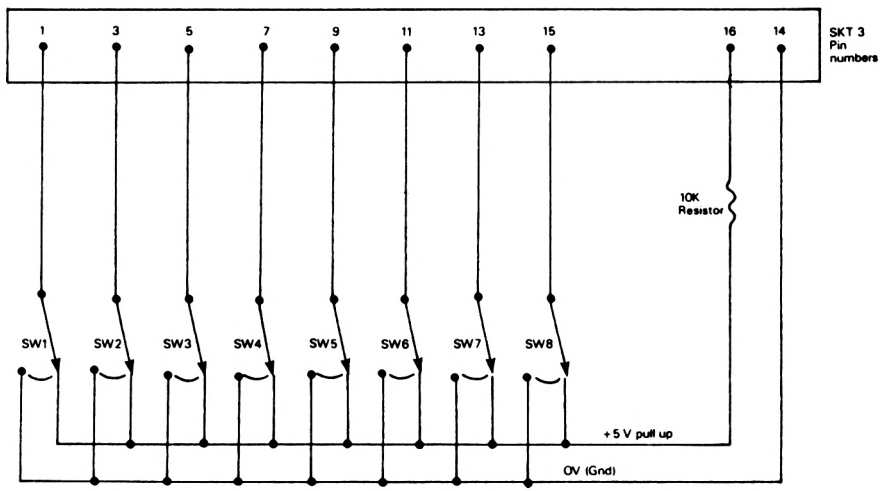


Fig 3.35 Esquema de la caja de pulsadores para el registro de sensores

Lo que necesitamos ahora es un programa que lea repetidamente el registro de sensores, y preparar los bucles de retardo adecuados cuando se pulse el interruptor. (Vea el texto del proyecto tres para obtener la explicación sobre estos bucles de retardo). Este programa debe contar el número de veces que se ha pulsado cada pulsador durante el tiempo que ha estado funcionando, y mantener en la pantalla los valores obtenidos. Una facilidad adicional puede ser almacenar en una cinta las estadísticas obtenidas, pulsando la tecla "#" en el teclado

del CPC. Esto nos permitirá tener un programa posterior que produzca un listado con los porcentajes obtenidos del contenido de estos ficheros. Los contadores deben ser de 16 bits, lo que significa que los valores a contar irán desde 0 a 65535. La figura 3.36 es el listado del programa BASIC que realiza este trabajo. Posteriormente, en el capítulo 5, volveremos a tratar esta aplicación, y desarrollaremos un programa en lenguaje ensamblador para realizar una función similar.

```
100   ' **        Registro de datos - versión BASIC                **
      ' **                                                         **
200   ' ** El registro de datos cuenta las pulsaciones efectuadas en **
        ** cada uno de los pulsadores conectados al registro de     **
300   ' ** sensores, detallado como proyecto 'hardware' en el       **
        ** capítulo tres de este libro.                            **
400   CLEAR: MODE 2: FOR I=0 TO 7: LOCATE ((10*I)+1),1: READ T$:
      PRINT T$;: NEXT: OUT &F8F7,&AF: PASS=1: DIM C(20)
500   ' ** La línea 400 imprime los tipos de aves que podemos      **
        ** encontrar, pero se puede cambiar por cualquier otra cosa **
600   ' ** que se desee como, tipos de coches, tipos de gente, etc. **
        ** Ahora podemos explorar los pulsadores repetidamente e    **
      ' ** incrementar el contador de la matriz C que está asociada **
700   ' ** con el pulsador. Si el usuario pulsa la tecla "#", se     **
        ** vuelcan los datos a un fichero en disco o cinta.         **

800   R=0: I$="": WHILE R=0 AND I$<> "#": R=INP(&F8F5): I$=INKEY$:
      GOSUB 5000: WEND: IF I$="#" THEN 1000 ELSE FOR I=0 TO 7:
      IF (R AND (2^I)) <> 0 THEN C(I+1)=C(I+1)+1
900   NEXT I: FOR DB=1 TO 50: NEXT: GOTO 800
      ' ** El bucle FOR-NEXT de DB es un bucle de retardo          **

1000  ' ** Rutina para escribir los datos a un fichero            **

1100  OPENOUT "DATLOG."+STR$(PASS): FOR I=1 TO 8: PRINT #9,STR$(C(I)):
      C(I)=0: NEXT: CLOSEOUT: PASS=PASS+1: LOCATE 1,2: PRINT CHR$(20);:
      GOTO 800 ' ** Creamos el fichero DATLOG.xx (donde xx es el   **
      ** número del ejemplo                                       **
5000  ' ** Pequeña rutina para actualizar el contador en pantalla  **

5100  FOR I=0 TO 7: LOCATE ((I*10)+1),2: PRINT C(I+1);: NEXT: LOCATE 1,5: RETURN
32000 DATA GORRION,LECHUZA,ABADEJO,CUERVO,HALCON,PETIRROJO,PINZON,JILGUERO
```

Fig 3.36 Programa BASIC para captura de datos

Conclusión

Hemos visto una aplicación particular del registro de sensores, la obtención de estadísticas. Este circuito básico de E/S tiene muchas más aplicaciones.

Detalles de construcción

Lo único que se puede decir sobre la construcción del registro de sensores es para repetir lo dicho anteriormente, No conecte nunca al PPI señales que excedan de los nivele lógicos normales. Si lo hace inadvertidamente, puede dañar permanentemente su CPC.

PROYECTO 6: CANAL DE TRASFERENCIA PARALELA

Este es otro proyecto que se construye en la tarjeta multi proyecto. Su propósito es permitir la trasferencia rápida de datos de un CPC a otro, o entre un CPC y otro ordenador.

Con la llegada de las redes de datos para ordenadores, se ha hecho normal el intercambio de programas y datos entre máquinas. Esto se usa más en el campo de los mini y micro ordenadores. Los usuarios de ordenadores personales han comenzado a interesarse por este tipo de trasferencia sobre redes de datos. El canal de trasferencia paralela de datos, que presentamos aquí, le permitirá conectar varias máquinas CPC en una pequeña red de datos local. Esto supone la construcción del circuito que presentamos, y también el programador de EPROM y la tarjeta de expansión de ROM. El programa que maneja la red de datos se presentará en el capítulo seis.

Veamos algunas ideas, sin desarrollar, para la utilización del canal de trasferencia paralela. Este canal nos proporciona una vía para trasferir datos, octeto a octeto, desde nuestro CPC a otro dispositivo que admita los datos en octetos completos. También nos proporciona la capacidad de trasferir datos a mayor velocidad de la que es posible con una conexión serie RS232. Por lo tanto, podemos hacer trasferencia masiva de datos desde, o hacia el CPC, en lugar de limitarnos solamente a la trasferencia de mensajes cortos.

Una aplicación frecuente de la electrónica, en el campo científico, es un dispositivo de captura de datos. El dispositivo de captura es una caja negra, usada para registrar temperaturas, humedad o cantidad de luz solar, para estudios sobre el tiempo. Estos dispositivos suelen contener una memoria alimentada por baterías, o una perforadora de cinta, o una cinta magnética, donde almacenan los datos obtenidos.

El factor común de los dispositivos de captura de datos es que antes o después deben volver al laboratorio, aula u oficina. Suelen conectarse entonces a un ordenador para poder leer los datos, almacenarlos y analizarlos. El canal de trasferencia paralela se puede usar para introducir este tipo de datos dentro del CPC.

Otra posible aplicación es conectar impresoras. Existen muchas impresoras que no se pueden conectar a la puerta de impresora del CPC, porque necesitan códigos de carácter de ocho bits, mientras que la puerta de impresora del CPC solo trasfiere 7. (Ver apéndice 7).

También es posible la trasferencia de datos en paralelo desde ordenadores grandes al CPC. Todos los fabricantes de mini y micro ordenadores fabrican también algún interfase paralelo de datos, que se puede conectar al que presentamos aquí. Si tiene acceso a una máquina grande, y puede conseguir un interfase de este tipo, podrá producir cartas, programas y datos en su CPC, y conectarlo después al ordenador grande para trasferir su trabajo rápidamente. No creo que ningún jefe rechace este tipo de trabajo en casa. Si demuestra que la idea funciona, puede conseguir un CPC conectado permanentemente al ordenador grande. En comparación con el precio de un miniordenador o uno grande, el precio de un segundo CPC resulta minúsculo, por lo que puede ser una buena solución para un gran compañía.

'HARDWARE' DEL CANAL DE TRASFERENCIA PARALELA

Una vez más, el proyecto depende del chip PPI. Aparte del decodificador de direcciones y del PPI, este es realmente un proyecto 'software'. Para el canal de trasferencia paralela se usan las puertas A, B y la mitad de la C, del PPI #1. Este PPI funciona en modo 1, con un octeto de control igual a hex AF. Con esto preparamos la puerta A como puerta de salida de 8 bits y la puerta B como puerta de entrada de ocho bits. Los bits 6 y 7 de la puerta C se usan como señales de estado de la puerta A, y los bits 1 y 2 de la puerta C, como señales de estado de la puerta B. Los bits 0 a 3 de la puerta C permanecen disponibles para poder usarlos con interrupciones, pero no se usan en este diseño. Los bits 4 y 5 de la puerta C se preparan como entradas, para usarlos en los proyectos del sintetizador de palabra y el de la matriz de teclas, que hemos descrito anteriormente en este capítulo. El PPI #1 y sus circuitos lógicos y conectores asociados, se muestran en la figura 3.37.

Refiriéndonos a la figura 3.37, el PPI #1 aparece con todas las líneas requeridas para el canal de trasferencia paralela, conectadas al conector PLG1, que es un conector tipo 'D' de 25 contactos. El listado de las conexiones del conector PLG1 se dará después de explicar el significado de cada una de las señales. E4a es un circuito inversor que invierte la señal de RESET del 'bus' de expansión del CPC. Se hace así porque el 8255 necesita una señal de nivel lógico bajo para su entrada RESET, y solo se debe poner alta cuando se tiene que restaurar el chip. La entrada CS negativa del PPI #1 viene desde la salida YO del decodificador de direcciones (refiérase a la figura 3.15). Esto nos pone el PPI en las siguientes direcciones:

```
Puerta A              Hex F8F0
Puerta B              Hex F8F1
Puerta C              Hex F8F2
Registro de control   Hex F8F3
```

Por lo tanto, el comando BASIC:

```
OUT &F8F3,&AF
```

inicializará el PPI de la forma que se ha detallado en el primer párrafo de esta sección.

El modo de operación seleccionado en el PPI, es el ideal para la trasferencia de datos en paralelo. Los dos dispositivos intercomunicados se conectan mediante un cable de 25 hilos. Puede ser del tipo cinta, pero dará mejor resultado un cable apantallado convencional. La longitud máxima de esta interconexión no debe exceder de los 9 metros (30 pies), en un entorno favorable (esto es, que el cable no vaya paralelo a las líneas de corriente eléctrica o cerca de generadores de ruidos eléctricos, como trasformadores o motores), En condiciones adversas se puede considerar un máximo de 3 metros (10 pies). Al final de esta sección se proporciona un programa de prueba, junto con el circuito para probar el canal de trasferencia paralela.

La puerta B de entrada, tiene 8 líneas de entrada de datos, junto con otras dos señales adicionales. Estas señales son la de sincronismo de entrada (strobe) y la línea de salida de 'memoria intermedia llena' (IBF). La salida de sincronismo se activa con nivel cero, cuando el procesador ha leído los datos que había a la entrada. Una

vez que ha sucedido esto, se puede trasferir el siguiente octeto.

La puerta A de salida, tiene 8 líneas de salida de datos, y dos líneas de estado, asociadas. Estas dos líneas son, la salida de 'memoria intermedia de salida llena' (OBF negativa), y la línea de entrada ACK negativa. La señal OBF negativa se pone a cero lógico cuando el procesador ha cargado un octeto de datos dentro de la puerta de salida, la línea ACK negativa, la pone a cero lógico el dispositivo remoto, para decirle a la puerta que ya ha leído los datos que le ha presentado. La figura 3.38 muestra cómo se hace la conexión práctica de los dos canales de trasmisión paralela. Sin embargo, una explicación 'humana' de la secuencia de trasferencia nos aclarará el uso de estas señales:

STROBE (sincronismo): (desde el dispositivo remoto) Cuando se pone a cero significa - "Le digo al amigo CPC que acabo de dejar un octeto en su puerta de entrada".

IBF: (desde el CPC) Cuando se pone a uno significa - "¡Ah! ya lo has dejado, espera un momento mientras encuentro un lugar donde almacenarlo".

OBF: (desde el CPC) Cuando se pone a cero significa - "¡Eh Dispositivo remoto!, He dejado un octeto en mi puerta de salida para que lo recojas".

ACK: (desde el dispositivo remoto) - "Ya he recogido tu octeto y lo he guardado".

Como mencionamos antes brevemente, cuando el PPI se programa para operar con esta parte de la puerta C como señales de control, estas señales son:

Puerta C bit 7 = OBF negativo
Puerta C bit 6 = ACK negativo
Puerta C bit 2 = STB negativo
Puerta C bit 1 = IBF
Puerta C bits 3 y 0 = se pueden usar para señales de interrupción, no se usan en este proyecto.

La ventaja de usar el 8255 en esta forma es que el procesador queda liberado de cualquier responsabilidad en la trasferencia de datos, ya que el 8255 se encarga de todo el protocolo entre él mismo y el dispositivo remoto. (Protocolo es el término técnico para la secuencia de señales necesarias para sincronizar la trasferencia de datos entre dispositivos conectados a ordenadores). Todo lo que usted necesita es escribir dentro de la puerta A el octeto que va a mandar, y ejecutar un bucle de retardo hasta que la señal OBF se ponga positiva. Para introducir dentro del procesador el octeto trasmitido, solo necesita comprobar el estado de IBF (Puerta C bit 1). Si está a uno, el octeto mandado por el dispositivo remoto está disponible en la puerta B.

Ahora que ya hemos visto todas las señales involucradas, aquí tenemos una lista de las señales disponibles en el conector PLG1, que se ve en el esquema de la figura 3.37.

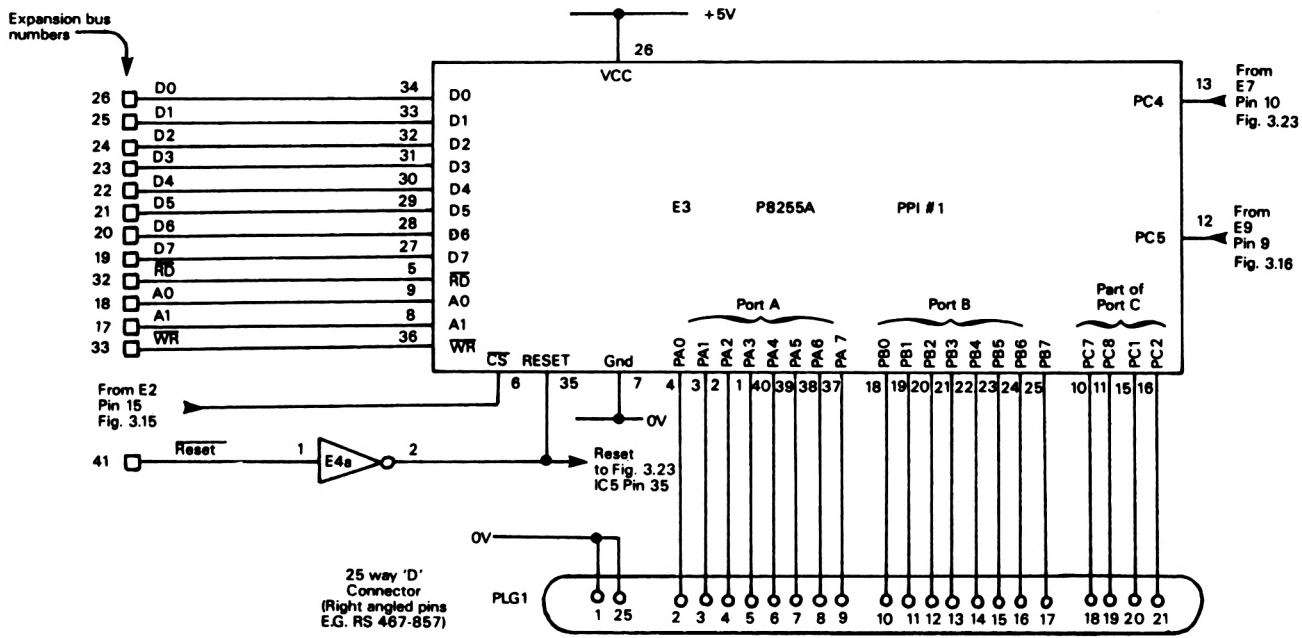


Fig. 3.37: PPI #1 Esquema del canal de transferencia paralela.

Uso de las patillas del conector PLG1

Patilla	Nombre de la señal	Patilla	Nombre de la señal
1	0 voltios	14	Bit 4 dato entrada
2	Bit 0 dato salida	15	Bit 5 dato entrada
3	Bit 1 dato salida	16	Bit 6 dato entrada
4	Bit 2 dato salida	17	Bit 7 dato entrada
5	Bit 3 dato salida	18	OBF negativo (PC7)
6	Bit 4 dato salida	19	ACK negativo (PC6)
7	Bit 5 dato salida	20	IBF (PC1)
8	Bit 6 dato salida	21	STB negativo (PC2)
9	Bit 7 dato salida	22	No se usa
10	Bit 0 dato entrada	23	No se usa
11	Bit 1 dato entrada	24	No se usa
12	Bit 2 dato entrada	25	No se usa
13	Bit 3 dato entrada		

La figura 3.38 lista el programa BASIC para la comprobación de la parte electrónica del canal de trasferencia paralela, junto con la lista de las patillas del conector tipo D, hembra, para conectar la puerta de salida con la de entrada. Si después de hacer funcionar el programa, sin errores, durante unas dos horas, comprueba que hay corrupción de datos al conectarlo a otra máquina, esto le indicará que el cable es más largo de lo permitido, o que tiene cerca una fuente de ruido eléctrico. Si detecta que es siempre el mismo bit el que está en error, puede que tenga una soldadura mal hecha en PLG1 o que las conexiones no sean correctas. La solución consiste en comprobar las soldaduras, reconectar el cable, apantallarlo, si no lo está aún, o preparar un pequeño circuito regenerador en cada extremo (ej. TTL 74LS07 o 74LS128).

Detalles de los puentes en el conector para la prueba de PTC

Se deben puentear las siguientes patillas en un conector D, para hacer el conector de prueba requerido por el programa de comprobación que viene a continuación:

```
Conectar   2  con  10
Conectar   3  con  11
Conectar   4  con  12
Conectar   5  con  13
Conectar   6  con  14
Conectar   7  con  15
Conectar   8  con  16
Conectar   9  con  17
Conectar  18  con  21   (OBF negativo con STB)
Conectar  19  con  20   (IBF con ACK negativo)
```

Uniendo las patillas indicadas haremos un bucle entre el canal transmisor y el receptor, el programa mandará entonces todos los valores de caracteres posibles y se asegurará de que los recibe correctamente, con un informe de los errores que vayan apareciendo.

```
1000  MODE 2: HED$=STRING$(79,&2A): PRINT HED$: PRINT TAB(17);
      "Programa de prueba del canal de trasferencia paralela": PRINT HED$
2000  LOCATE 1,5: PRINT "Fase de preparación de la prueba": LOCATE 1,7:
      INPUT "Está conectado el conector de prueba en PLG1? <No>";R$:
      R$=UPPER$(LEFT$(R$,1)): IF R$ <> "S" THEN LOCATE 1,7 ELSE 2200
2100  PRINT "Se debe conectar el conector de prueba detallado en el libro ":
      PRINT "en el conector PLG1 de la tarjeta Multi Proyecto":
      GOTO 32767
2200  LOCATE 1,5: PRINT CHR$(20);"Ejecutando prueba del PTC": LOCATE 1,8:
      PRINT "La prueba ha comenzado: Pasada xx completa: Errores hasta
      ahora = xx "
2300  OUT &F8F3,&AF: FOR P=1 TO 999: FOR T=255 TO 0 STEP -1:
      OUT &F8F0,T: WHILE (INP(&F8F2) AND 2) = 0: WEND: R=INP(&F8F1):
      IF R <> T THEN GOSUB 10000
2400  NEXT T: LOCATE 32,8: PRINT P;: NEXT P: GOTO 32767
10000 LOCATE 1,17: PRINT CHR$(7);HED$: PRINT TAB(20);"Informe de error":
      PRINT HED$: PRINT: PRINT "Mandado Hex ";HEX$(T);": Recibido Hex ";
      HEX$(R): PRINT: BITERR= (T XOR R)
10010 PRINT TAB(50);"Mandado ";SPACE$(8-LEN(BIN$(T)));BIN$(T):
      PRINT TAB(50);"Recibido ";SPACE$(8-LEN(BIN$(R)));BIN$(R)
10015 LOCATE 1,22: PRINT "Bits en error:   ";: FOR B=0 TO 7:
      IF (BITERR AND (2^B)) <> 0 THEN PRINT B;";";
10020 NEXT: FAILS=FAILS+1: LOCATE 67,8: PRINT FAILS: LOCATE 1,25:
      PRINT "Pulse una tecla para continuar";: R$="": WHILE R$="":
      R$=INKEY$: WEND:  LOCATE 1,17:  PRINT CHR$(20);:  RETURN 32767  LOCATE 1,20:
PRINT "Fin de la ejecución"
```

Fig 3.38 Programa de prueba del canal de trasferencia paralela

'Software' del canal de trasferencia paralela

El 'software' para poder usar el canal de trasferencia paralela en una red local de datos, se detallará en el capítulo seis. Por ahora incluimos el listado de dos programas en código máquina que realizan las operaciones fundamentales del canal. (figuras 3.39 y 3.40). Como aún no hemos tratado la programación en código máquina, puede que no entienda estos dos programas, así que marque estas páginas, para volver a ellas más adelante. El programa de trasmisión de la figura 3.39 es una subrutina que manda datos con la longitud indicada por el valor que aparece en el par de registros DE, y la dirección de comienzo de los datos en el par de registros HL. El programa de recepción listado en la figura 3.40, es una subrutina que toma algunos argumentos de los mismos registros, pero los usa como parámetros para saber cuantos octetos va a recibir. Si pulsa CTRL/A durante la ejecución, se interrumpe la recepción.

La figura 3.34 es el programa BASIC para recibir los datos y grabarlos en cinta o en la pantalla. Tenga cuidado de no escribir sobre la pantalla los datos tal y como los recibe, puede obtener resultados muy extraños, y perder el programa, así que salve primero el programa. Finalmente, la figura 3.42 es el programa BASIC para obtener los datos del cassette o del teclado y mandarlos al canal de trasferencia paralela.

```
100 ;
110 ;
120 ; ** SEND. Esta subrutina en código máquina manda un número de  **
130 ; ** octetos sobre el canal de trasferencia paralela (PTC).    **
140 ; ** A la entrada, el registro HL apunta al primer octeto a    **
150 ; ** mandar, y DE contiene el número de octetos que se van a   **
160 ; ** mandar. AF no es válido a la vuelta.
170 ;
180 PTCSND:   EQU  #F8F0  ; Define la dirección de los registros PTC
190 PTCREC:   EQU  #F8F1
200 PTCCNT:   EQU  #F8F2  ; Señales de control para los canales
210 PPICTL:   EQU  #F8F3  ; Registro de control del chip PPI del PTC
220 ;
230 ;
240 ;
250 ;
260           ENT  $           ; Aquí comienza la ejecución
270 SEND:     PUSH BC
280           PUSH DE
290           PUSH HL
300 SEND00:   LD   BC,PTCCNT   ; BC apunta al registro de estado del PTC
310           IN   L,(C)  ; Estad del PTC en L
320           BIT  7,L         ; Ver si el canal está preparado para mandar
330           JR   NZ,SEND01   ; Si, saltar
340           CALL #BB09  ; Ver si se ha tecleado algo
350           JR   NC,SEND00   ; No, comenzar de nuevo
360           SCF              ; Si, poner bit de acarreo
370           JR   SEND04 ; y terminar
380 SEND01:   LD   BC,PTCSND   ; BC apunta al canal trasmisor
390           LD   A,(HL) ; A=octeto a mandar
400           OUT  (C),A  ; mandarlo
410           LD   A,#0   ; Ver si está a cero el contador
420           CP   E           ; comparar con E
430           JR   NZ,SEND05   ; octeto inferior <> 0, repetir bucle
440           CP   D           ; comparar con D
450           JR   Z,SEND06    ; si DE=0000, terminar
460 SEND05:   DEC  DE          ; si DE<>0, restarle uno
470           INC  HL          ; incrementar apuntador datos
480           JR   SEND00 ; repetir el bucle
490 SEND06:   AND  A           ; Poner bit de acarreo a cero
500 SEND04:   POP  HL          ; POP de la pila
510           POP  DE
520           POP  BC
530           RET              ; retornar
540 ;
550 ;
```

Fig 3.39 Programa en C/M para trasmitir desde el PTC

```
100 PTCSND:   EQU  #F8F0  ; Define la dirección de los registros PTC
110 PTCREC:   EQU  #F8F1
120 PTCCNT:   EQU  #F8F2  ; Señales de control para los canales
130 PPICTL:   EQU  #F8F3  ; Registro de control del chip PPI del PTC
140 ;
150 ;
160 ; ** RX: Esta rutina recibe los octetos del canal de trasferencia    **
```

```
170 ; ** paralela (PTC), los coloca en una memoria intermedia, que está **
180 ; ** apuntada a la entrada por el registro HL, y devuelve control    **
190 ; ** cuando ha recibido el número de octetos indicado a la entrada   **
200 ; ** por el registro DE. Cuando el bit uno de PCTCNT se ponga a uno, **
210 ; ** es que hay un octeto disponible. Mientras RX está esperando,    **
220 ; ** llama también a una rutina de la ROM para comprobar si se ha    **
230 ; ** pulsado CTRL/A. Si es así, RX devuelve el bit de acarreo puesto **
240 ; ** El registro AF no contiene datos válidos al retorno            **
250 ;
260 ;
270 ;
280             ENT     $                   Empieza    la    ejecución    aquí
290 RECEEV:   PUSH BC
300           PUSH HL
310           PUSH DE
320 RECEE2:   LD   BC,PTCCNT   ; BC apunta al canal de control
330           IN   A,(C)  ; Obtiene estado del canal
340           BIT  1,A          ; Ver si IBF es uno
350           JR   NZ,RECEE1    ; si, saltar
360           CALL #BB09  ; comprobar si CTRL/A
370           JR   NC,RECEE2    ; no, bucle
380           CP   #1           ; ver si CTRL/A
390           JR   NZ,RECEE2    ; no, bucle
400           SCF               ; poner bit acarreo
410           JR   RECOUT ; terminar
420 RECEE1:   LD   BC,PTCREC    ; BC apunta a canal receptor
430           IN   A,(C)  ; Tomar un octeto
440           LD   (HL),A ; almacenarlo
450           LD   A,#0   ; poner a cero el registro A
460           CP   E            ; ver si ha terminado
470           JR   NZ,RECEE3    ; no, repetir bucle
480           CP   D            ; ver si ha terminado
490           JR   NZ,RECEE3    ; no, repetir bucle
500           AND  A            ; limpiar bit acarreo
510           JR   RECOUT ; terminar
520 RECEE3:   DEC  DE           ; Decrementar contador octetos
530           INC  HL           ; incrementar apuntador datos
540           JR   RECEE2 ; repetir bucle
550 RECOUT:   POP  DE
560           POP  HL
570           POP  BC
580           RET               ; retornar
590 ;
```

Fig 3.40 Programa BASIC para recibir datos del PTC

```
100   '** Programa para recibir caracteres desde el PTC y
      ** ponerlos en un fichero en cinta o disco
110   MODE 2: INPUT "Datos a pantalla o a cinta? "; R$: R$=UPPER$(LEFT$(R$,1)):
      IF R$="P" THEN S=0 ELSE S=9
120   IF S=9 THEN INPUT "Nombre del fichero en disco o cinta ";F$:
      IF INSTR(F$,".") <> 0 THEN 120 ELSE OPENOUT F$:
      PRINT "Fichero abierto"
130   ON BREAK GOSUB 150
```

```
140   WHILE (INP(&F8F2) AND 2) = 0 : WEND: PRINT #S,CHR$(INP(&F8F1));:
      GOTO 140
150   CLOSEOUT: IF S <> 0 THEN PRINT "Recepción abortada - fichero ";
      F$;" creado"
32767 END
```

Fig 3.41 Programa BASIC para recibir datos y escribirlos a cinta

```
1000  '** Programa para mandar datos al PTC desde cinta o memoria
1010  ON BREAK GOSUB 10000: MODE 2: INPUT "Fuente de los datos: (T)eclado o
      (F)ichero";R$: R$=UPPER$(LEFT$(R$,1)): IF R$="F" THEN 2000 ELSE IF R$="T"
      THEN 3000 ELSE 1010
2000  CLS: CAT: INPUT "Que fichero tengo que mandar? ";FF$:
      INPUT "Tipo de fichero - Bimario o ASCII";T$: T$=UPPER$(LEFT$(T$,1)):
      IF T$="A" THEN 2010 ELSE 2100
2010  OPENIN FF$: WHILE EOF=0: LINE INPUT #9,R$:
      FOR I=1 TO LEN(R$): WHILE (INP(&F8F2) AND &80) = 0: WEND:
      OUT &F8F0,ASC(MID$(R$,I,1)): NEXT: WEND: GOTO 10000
2100  MEMORY &3FFF: LOAD FF$,&4000: PRINT "Se mandarán 16K desde Hex 4000":
      FOR I=1 TO 16384: WHILE (INP(&F8F2)) AND &80) = 0: WEND:
      OUT &F8F0,PEEK(&4000+I): NEXT: GOTO 10000
3000  CLS: PRINT "Teclee los caracteres a mandar y termine con ESC":
      FOR I=1 TO 65000: X$="": WHILE X$="": X$=INKEY$: WEND: PRINT X$;:
      OUT &F8F0,ASC(X$): NEXT
10000 CLS: PRINT "Terminada transferencia al PTC"
32767 END
```

Fig 3.42 Programa BASIC para mandar datos desde cinta o teclado al PTC

Condensadores

C1	50 mfd 16V electrolítico
C2-C3	0.1 mfd 16V disco cerámico

Semiconductores

E1	TTL 74LS20 2 puertas NAND 4 entradas
E2	TTL 74LS138 Decodificador de 3 a 8 líneas
E3	8255A chip PPI
E4	TTL 74LS04 inversores Hex

Varios

Zócalos para todos los chips
PLG1 conector D de 25 patillas en ángulo
Conector para el 'bus' de expansión Circuito impreso

Fig 3.43 Lista de componentes para el PTC

Esta lista de componentes es para todos los proyectos de la tarjeta multi proyecto. Si solo quiere construir alguno de los proyectos, refiérase a las listas individuales de componentes que vienen con cada proyecto.

Condensadores

C1	50 mfd	16V	electrolítico
C2-C5	0.1 mfd	16V	disco cerámico
C6	27 pf	16V	poliestireno
C7	0.1 mfd	16V	cualquier tipo
C8-C9	0.22 mfd	16V	

Resistencias

R1-R8	10K Ohmios	1/4 watio
R9	470 Ohmios	1/4 watio
R10-R11	10K Ohmios	1/4 watio
RV1	47K potenciómetro para circuito impreso	

Semiconductores

E1	TTL 74LS20 2 puertas NAND 4 entradas
E2	TTL 74LS138 Decodificador de 3 a 8 líneas
E3	8255A chip PPI
E4	TTL 74LS04 inversores Hex
E5	8255A chip PPI
E6	74154 decodificador de 4 a 16 líneas
E7	74150 selector de datos de 16 entradas
E8	TTL 74LS32 4 puertas OR de 2 entradas
E9	SP0-256
E10	TTL 74LS174 básculas de datos Hex
E11	TTL 74LS629 oscilador controlado por tensión

Varios

Zócalos para todos los chips

PLG1	Conector D de 25 patillas en ángulo
SK1	Zócalo DIL de 40 patillas
SK2	Zócalo DIL de 14 patillas
SK3	Zócalo DIL de 16 patillas

Conector para el 'bus' de expansión Circuito impreso

Fig 3.44 Lista completa de componentes para la tarjeta multi proyecto

Notas de construcción

Si está usando la tarjeta multi proyecto, no debe tener dificultad con éste, la única dificultad se le puede presentar al construir el cable para conectar el canal de trasferencia paralela a otro dispositivo. Aún esto no representará graves problemas, ya que en los conectores vienen marcados los números de las patillas, y por lo tanto, el único problema será la longitud del cable y el camino que se le va a hacer seguir. Como se dice en la lista de componentes de la figura 3.43, el 8255 debe ser de la versión "A", he comprobado que muchos chips 8255 no funcionan satisfactoriamente. Use siempre zócalos para todos los chips.

Conclusión

El canal de trasferencia paralela representa un forma de expandir sus actividades informáticas. Esto se debe a que le permite intercambiar datos con otros ordenadores cercanos, aunque no por las líneas de comunicación (vea el siguiente proyecto). También le permite recibir datos desde unidades de captura de datos, y desarrollar sus propias ideas con equipos operados con baterías que pueden proporcionar datos a su CPC para su análisis.

Conclusión de la tarjeta multi proyecto

Con esto terminamos los proyectos que lleva la tarjeta multi proyecto. La figura final - 3.44 - es una lista completa de componentes para esta tarjeta.

PROYECTO 7: VERSIÓN "B" DEL INTERFACE RS232

En el proyecto seis vimos una forma de trasferir datos rápidamente entre las máquinas CPC y otros ordenadores o dispositivos. Como ya mencionamos en la descripción de ese proyecto, el cable de interconexión entre las dos máquinas está sujeto a problemas de ruidos, y como las señales son las estándar de niveles lógicos TTL, tiene limitada la distancia sobre la que se pueden trasmitir los datos. Puede conseguir trasmitir desde una habitación a otra, pero no a más distancia, y esto con mucha suerte.

Este proyecto elimina los problemas asociados con la trasmisión de datos sobre grandes distancias a niveles TTL. Si ha leído el apéndice 6, habrá comprendido porqué usamos conexiones RS232 entre ordenadores y terminales. La principal razón es que las señales viajan a través de los cables de conexión a niveles dos o tres veces superiores a las tensiones TTL, lo que aumenta la inmunidad a los ruidos. Otra razón es que hace tiempo que la industria de los ordenadores hizo del RS232 el principal interfase estándar para la trasmisión serie de datos. Esto hace que la mayoría de los ordenadores, ya sean grandes ordenadores u ordenadores personales, tengan disponible esta opción extra.

Veamos los inconvenientes del uso del RS232. Realmente no son muy severos. El primero es que los octetos se mandan (generalmente) por una línea de tres conductores: uno para mandar datos: uno de masa: uno para recibir datos. Esto hace que los circuitos que componen el interface serie tengan que serializar los datos, y cuando llegan al otro extremo, los tenga que deserializar, esto es, volverlos a convertir en octetos desde las series de impulsos que llegan. Esto no representa un gran problema, ya que hay muchos chips disponibles para realizar esta función por usted. El mayor problema es el tiempo, se tarda más tiempo en trasferir un número dado de octetos a través de tres hilos en una conexión serie que a través de una conexión paralela, y siempre será así, la tecnología no lo puede cambiar. La otra desventaja es que se necesitan más componentes electrónicos para implementar un esquema serie, pero podemos resignarnos.

Una vez completado este proyecto, nos proporcionará cuatro canales de ocho bit RS2323 para su CPC. ¿Para qué necesitamos cuatro canales? Usted probablemente no los necesitará. Se han incluido cuatro canales porque una vez que hemos construido el decodificador de direcciones,

y hemos conseguido los chip de trasmisión y recepción RS232 y el generador de velocidad de trasmisión, es muy sencillo añadir un chip tranmisor/receptor para otro canal. Aunque el diseño permite un máximo de cuatro canales, usted puede instalar solamente uno si quiere.

¿Qué usos puede tener este proyecto? Puede ser más rápido listar los usos para los que NO sirve . No sirve para hacer el café, o para hacer la cama, o para lavar la ropa. (¿No tiene algún amigo escéptico que le haya preguntado alguna vez si su CPC puede lavar la ropa?) Bueno, pongámonos serios, he aquí algunas de las cosas puede hacer:

1) Le permite usar el CPC como un terminal serie. De forma que si consigue un MODEM, puede conectarse a alguna red de datos.

2) Le permite usar su CPC para la preparación de datos, cuando los tenga listos puede trasmitirlos a otra máquina más grande.

3) Usando el 'software' que le presentamos más adelante en esta sección, puede conectar su CPC como una VDU para cuatro sistemas, con selección de funciones mediante el teclado, como una VDU real.

4) Puede conectar directamente otro CPC equipado de manera similar, u otro microordenador. Esto le permitirá tener un circuito cerrado de diálogo con otros usuarios en el mismo edificio.

5) Puede usar el CPC como herramienta de pruebas para impresoras o VDU en el equipo de un taller. Para esto deberá escribir algunos programas para mandar determinados datos - no es difícil.

Y hay muchas más aplicaciones.

El primero de los esquemas de la versión "B" del interface RS232 está en la figura 3.45. Esta figura nos muestra el decodificador de direcciones y el generador de velocidades de trasmisión y los circuitos de enclavamiento.

Aunque se muestran en otro esquema, cada canal tiene un chip llamado USART (Transmisor Receptor Asíncrono/Síncrono Universal). Se le suele conocer como chip PCI (Interface Programable de Comunicaciones). Nosotros usaremos el nombre de USART. Cada USART tiene un registro de control y un registro de datos. Cuando analicemos el segundo esquema, veremos este chip con más detalle.

Volvamos al circuito decodificador de direcciones que se muestra en la figura 3.45. La dirección base del interface es la dirección de E/S Hex F9E0. El interface ocupa las siguiente direcciones de E/S:

F9E0	USART1 registro de datos
F9E1	USART1 registro de control
F9E2	USART2 registro de datos
F9E3	USART2 registro de control
F9E4	USART3 registro de datos
F9E5	USART3 registro de control
F9E6	USART4 registro de datos
F9E7	USART4 registro de control
F9E8-B	Registro del generador de velocidad, canales 1 y 2
F9EC-F	Registro del generador de velocidad, canales 3 y 4

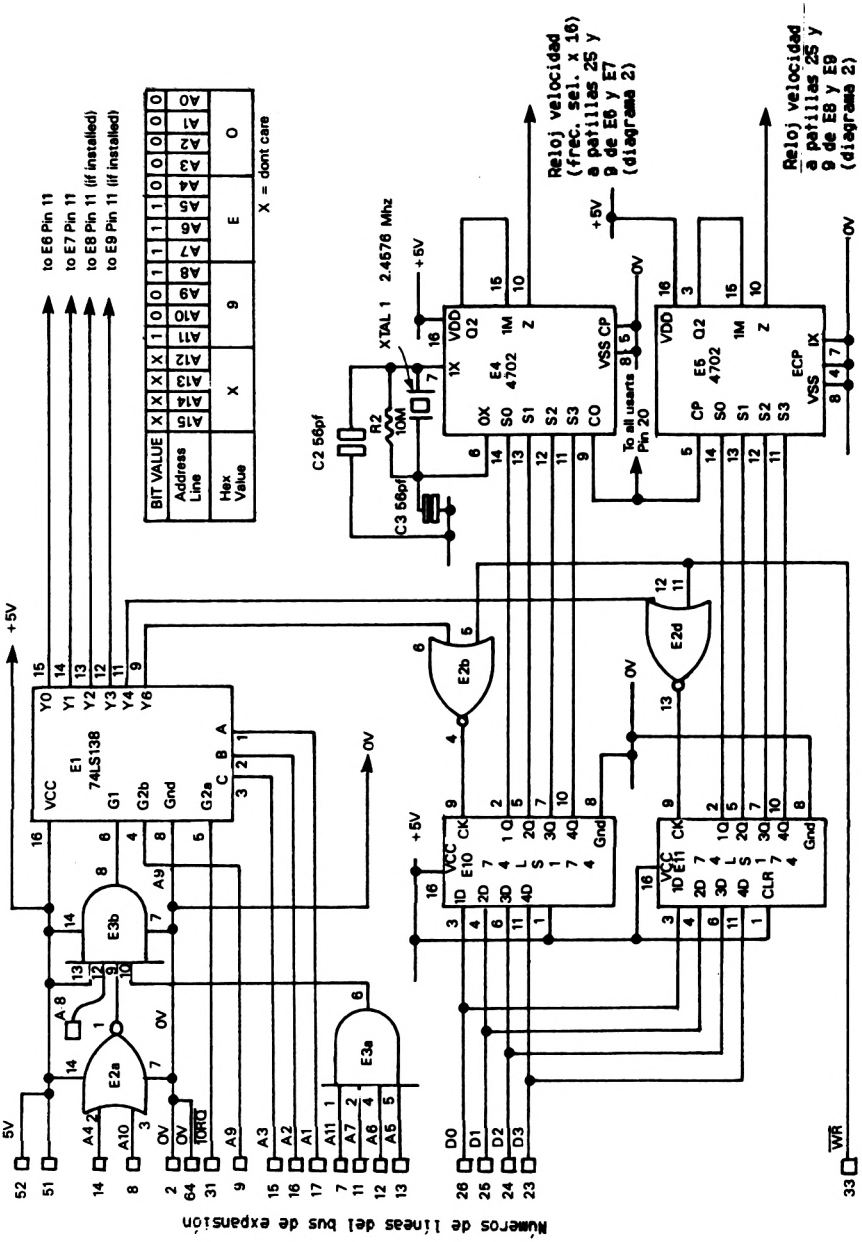


Fig 3.45 Esquema del decodificador de direcciones del RS232 VB

En la figura 3.45 se ve la máscara de bits para F9E0. Las puertas E2a, E3a y E3b activan a E1 (un 74LS138), cuando ocurre simultáneamente, una dirección que empiece por F9E, el sincronismo IORQ negativo y un cero en la línea de dirección A9. E1 decodifica entonces las líneas de dirección A1, A2 y A3 en uno de los seis sincronismos posibles del chip. Las puertas NOR E2b y E2d, mandan los sincronismos a los generadores de velocidad de trasmisión que, junto con el sincronismo de escritura, producen las señales de reloj para la carga de los circuitos de enclavamiento. Los registros de velocidad de transmisón son solo de escritura. Las otras cuatro señales de selección de chip van a parar a los USART, con la línea A0 para seleccionar los registros de control o de datos en el USART seleccionado.

Volviendo a los registros de selección de velocidad de trasmisión, el valor que ponemos en estos registros determina la velocidad seleccionada para cada pareja de chips USART. Los chips generadores de velocidad de trasmisión son del tipo 4702, y tienen sus entradas de selección conectadas a cuatro salidas de los registros de selección de velocidad. El valor colocado dentro de los registros corresponde a una velocidad de acuerdo a la siguiente tabla:

Valor Hex en el registro	*Velocidad seleccionada*
0	19200
1	19200
2	50
3	75
4	134.5
5	200
6	600
7	2400
8	9600
9	4800
A	1800
B	1200
C	2400
D	300
E	150
F	110

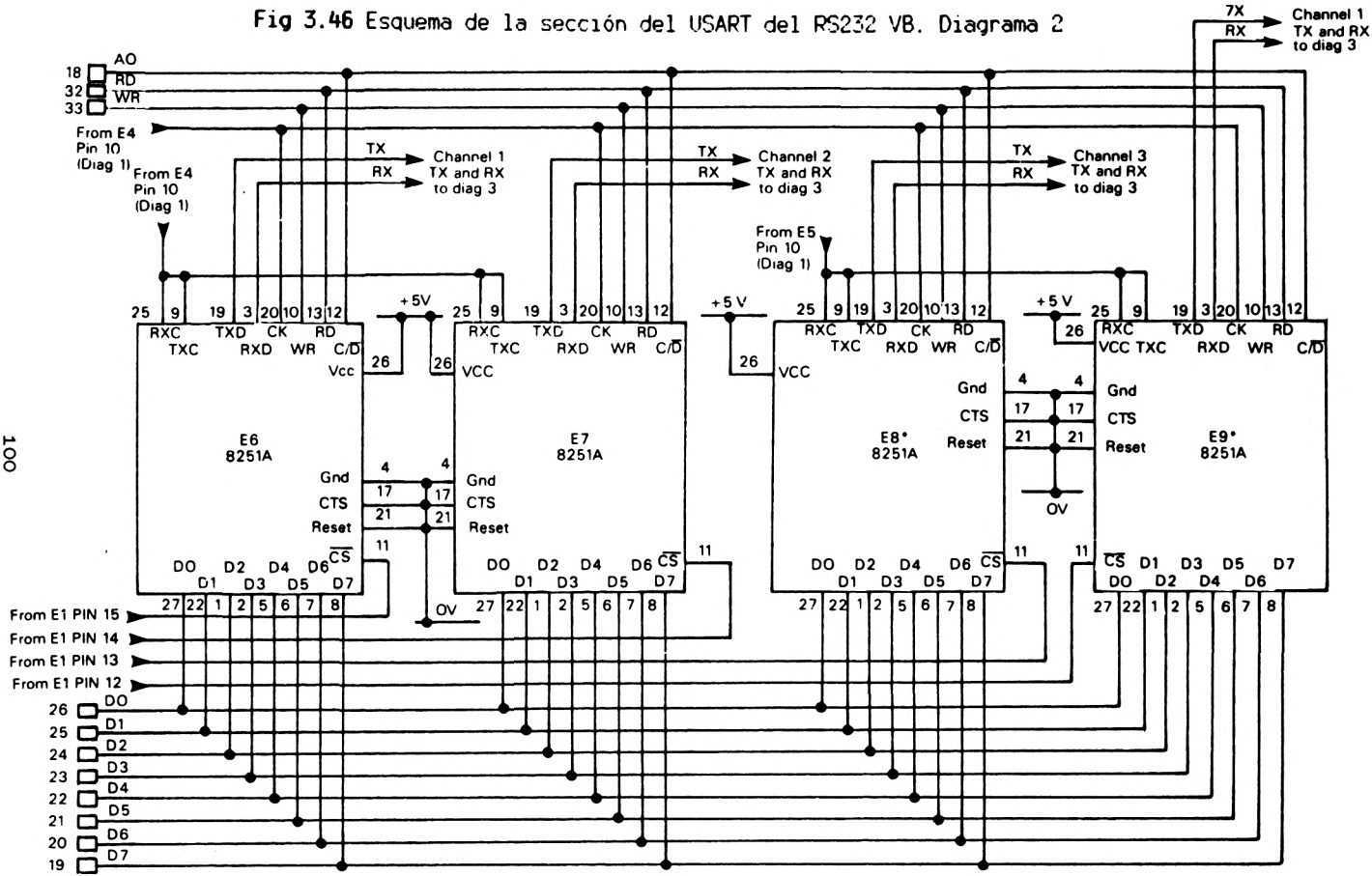
Así, por ejemplo, el comando directo:

```
OUT &F9E8,&D
```

Pone la salida de generador de velocidad de trasmisión de los canales dos y tres a 300 baudios. Realmente es un poco más complicado que esto, porque el chip generador de velocidad de trasmisión produce un frecuencia que es dieciséis veces mayor que la velocidad real. Esto se hace para compensar el hecho de que la mayoría de los chips USART dividen siempre por dieciséis la velocidad que reciben del reloj. Podemos programar el USART para que haga también esto, por lo que podemos ignorarlo, excepto para pruebas y para mirar con el osciloscopio.

La frecuencia maestra de los generadores de velocidad se obtiene del cristal XTAL1. El primer 4702 - E4 - usa el cristal y un circuito oscilador para producir una señal de reloj de 2.4576 MHz. Esta señal la usa E4 para producir las señales de reloj de velocidad de trasmisión, pero también se pasa a un segundo 4702 - E5. Esto evita tener que usar dos cristales y sus componentes asociados. La señal de reloj

Fig 3.46 Esquema de la sección del USART del RS232 VB. Diagrama 2



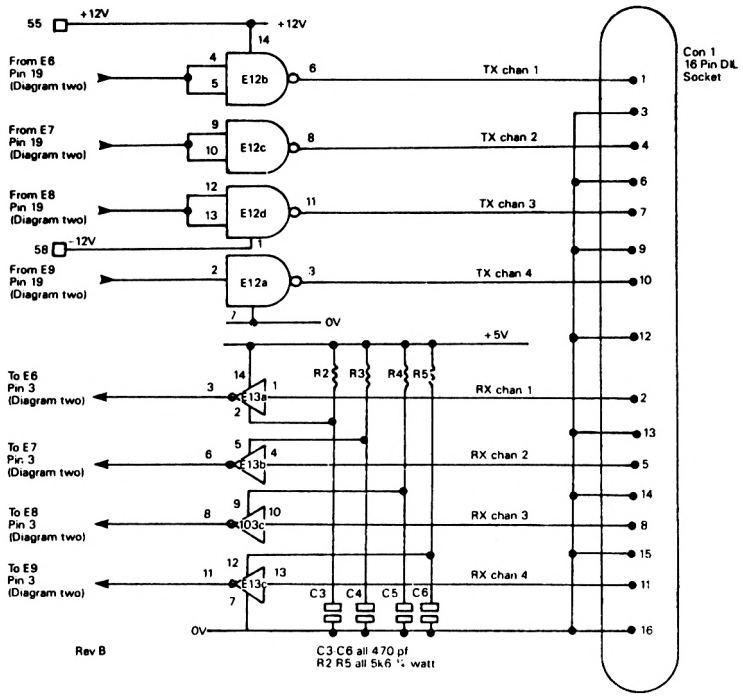


Fig 3.47 Sección del esquema del RS232 VB con los chips 1488 y 1489

seleccionada sale por la patilla 10 de cada 4702. El chip E4 alimenta a los USART uno y dos, mientras que E5 alimenta a los USART tres y cuatro. En la lista de componentes de este proyecto, que se da al final de la descripción del 'hardware' y del 'software', hay un pequeño diagrama de comprobación para ver los componentes que necesitará para un número de canales determinado.

La figura 3.46 muestra los USART. Todos ellos tiene sus líneas D0-D7 conectadas al 'bus' de datos. La entrada CS negativa (Selección de Chip activa baja) de cada USART se conecta desde el decodificador de direcciones como ya se ha descrito. Cada USART tiene su C/D negativo conectado a la línea A0. La línea C/D negativo decide cuando se accede al registro de control y cuando al de datos al poner la línea CS en estado bajo. Las señales de reloj de trasmisión y recepción se conectan al generador de velocidad E4 para los USART uno y dos, y a E5 para los USART tres y cuatro. No se ha suministrado forma de poder trabajar con velocidades distintas en trasmisión y recepción, ya que es raro el equipo que tenga esta facilidad, y además requeriría la inclusión de dos 4702 y dos circuitos de enclavamiento más. La patilla 20 de cada USART es su entrada para la señal del reloj maestro. Las especificaciones técnicas del 8251A indican que la frecuencia que aparece en esta patilla debe ser, al menos, treinta veces superior a

la que aparece en las entradas de reloj de trasmisión y recepción. Esto se consigue conectando las entradas de reloj maestro de todos los USART al reloj maestro del generador de velocidad de trasmisión, de 2.4576 MHz. Todos los USART se conectan a los sincronismos de escritura y lectura, y sus entradas RESET a masa.

La figura 3.47 muestra los circuitos RS232. Los dos chips usados son los 1488 y 1489. La red C/R en cada elemento del 1489 (E13) se ponen para mantener su frecuencia de respuesta a un límite útil, evitando el riesgo de que las señales de UHF causen recepciones espúreas. En esencia, el 1488 toma las señales a niveles TTL y las convierte en niveles RS232 (esto es, tensiones positivas y negativas para el cero y uno lógicos, respectivamente). El 1489 realiza la labor opuesta. Todas las líneas de trasmisión y recepción van a parar la conector CON1. Después de esto usted puede elegir el método de conexión que mejor le convenga, pero le sugiero un pequeño trozo de cable de cinta dividido en cuatro, soldado en conectores de tipo D.

Con esto damos por concluida la descripción del 'hardware' del interfase RS232 versión "B". Veamos ahora un programa para comprobar su funcionamiento, y otros programas para permitir el uso del Amstrad CPC como un terminal serie.

'SOFTWARE' PARA LA VERSIÓN B DEL INTERFASE RS232

Para usar el interfase se necesitan ciertos programas especiales. Esto es debido a que la ROM del CPC no está preparada para manejar un canal serie.

Antes de pasar a describir el 'software' que maneja el interface serie, el lector debería entender cómo opera el 8251A USART. El USART tiene dos registros. El primer registro es el de datos, en el que se introducen los octetos para trasmitirlos, y donde se reciben los octetos leídos de la línea serie. El segundo es el registro de control, que realmente tiene tres funciones. El octeto de modo solo se puede introducir dentro del registro de control nada más restaurar el 8251A, por medio de un impulso en su patilla RESET o mediante un RESET interno programado (que veremos más adelante). El octeto de modo selecciona las características operacionales del USART hasta el siguiente RESET. Se pueden seleccionar cosas como longitud de los datos en bits, paridad, y el número de bits de parada. Otra cosa que se puede efectuar mediante el octeto de modo es la modificación de la velocidad de trasmisión. Esto nos permite que dos USART conectados al mismo chip 4702 puedan operar con diferentes velocidades.

El diagrama del octeto de modo se puede ver en la figura 3.48. Como puede ver la señal de reloj de velocidad de trasmisión se puede dividir por: 1 (esto es, dejarla como está), por 16 (es la que se suele seleccionar, ya que el 4702 suministra 16 veces la velocidad seleccionada), o por 64 - que permite poner dos velocidades distintas a dos USART conectados al mismo 4702. Cuando programemos el octeto de modo deberemos consultar el manual del equipo al que nos vamos a conectar, o a su propietario. Como regla general, si no conoce qué características usar, inténtelo con ocho bits, sin paridad y con dos bits de parada. (Esta combinación es la que se suele llamar DEC estándar, debido a que la usan los mini ordenadores de Digital Equipment Corporation en la trasmisión serie). Si así no funciona y nadie

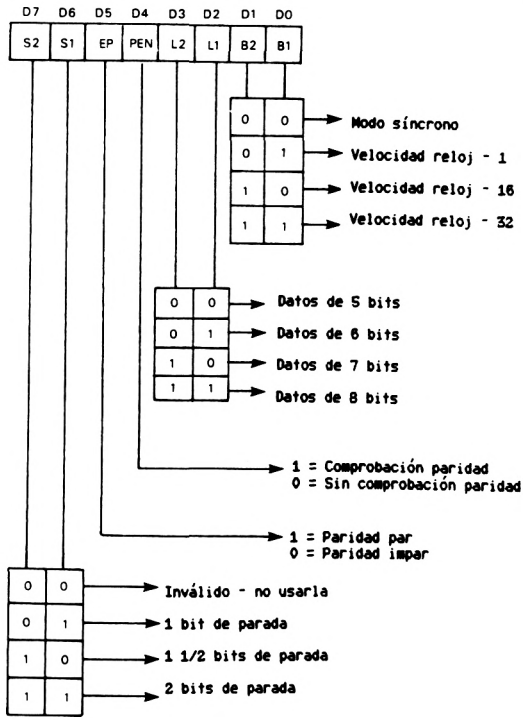


Fig 3.48 Diagrama del octeto de modo del chip 8251A USART

le puede proporcionar las características de trasmisión, me temo que la única forma de conocerlas es probando hasta que encuentre las correctas.

Los otros dos usos de la dirección del registro de control son, leer el estado del 8251A, y controlarlo. La figura 3.49 muestra la imagen en lectura del registro de control. Ahora vamos a ver la función de cada bit:

Bit 7:DSR (Grupo de Datos Preparado) - Muestra el estado de la patilla 22 del USART (Data Set Ready)

Bit 6:Detección de Sincronismo (Sync Detect) - se usa solamente en modo síncrono, por lo tanto no nos interesa ahora.

Bit 5:Error de Estructura (Framing Error) - Se pone cuando la lógica del USART detecta que el carácter que ha recibido tiene un número erróneo de bits de parada, debido normalmente a ruidos en la línea o a un octeto de modo incorrecto.

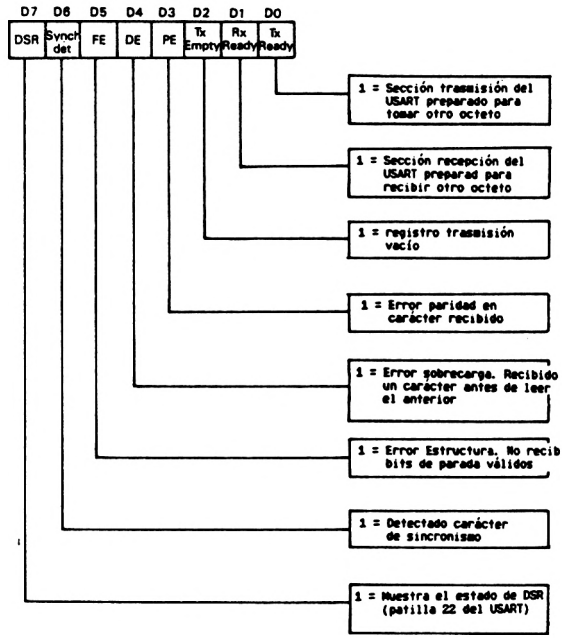


Fig 3.49 Imagen de lectura del registro de control del 8251A

Bit 2:TX vacío (TX empty) - Cuando está a uno, la memoria intermedia de trasmisión del USART - que tiene una longitud de dos octetos - está vacía.

Bit 1:RX preparada (RX ready) - Cuando está puesto, indica que se ha recibido un carácter, y que ya está disponible para ser leido en registro de datos.

Bit 0:TX preparada (TX ready) - Indica que la memoria intermedia de transmisión está preparada para admitir otro carácter para transmitirlo.

Ahora veamos la asignación de bits de la imagen de escritura del registro de control, que se muestra en la figura 3.50:

Bit 7:EH. (Enable Hunt) - Se usa solamente en modo síncrono, no lo usamos ahora.

Bit 6:IR. Internal Reset (Restauración interna) - Cuando pone a uno este bit, el USART será restaurado como si se hubiera aplicado un impulso a la patilla RESET, y el dispositivo tomará lo siguiente que se escriba en el registro de control como un octeto de modo.

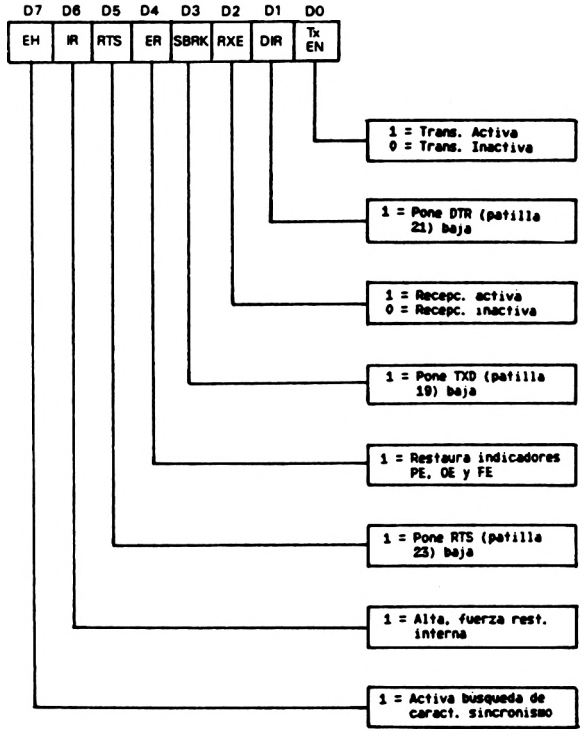


Fig 3.50 Imagen de escritura del registro de control del 8251A

Bit 4:Error de Sobrecarga (Overrun error) - Se pone cuando el carácter recibido no ha sido leído aún por el programa y llega otro a reemplazarlo - en otras palabras, se pierde un octeto.

Bit 3:Error de Paridad (Parity error) - La paridad recibida no coincide con la calculad sobre el octeto recibido. Suele deberse a ruidos en la línea, relojes de trasmisión inestables u octeto de modo incorrecto, que causan que el tranmisor y el receptor usen diferentes comprobaciones de paridad.

Bit 5:RTS - Aparecerá en la patilla 23 del USART el valor inverso del que se introduzca en este bit. (Si escribe un cero, la patilla RTS se pondrá a uno, y viceversa).

Bit 4:ER. Restauración de error (Error reset) - Se debe programar este bit después de detectar que se ha puesto uno de los bits de error. El hecho de que se haya puesto un bit de error no inhibe las operaciones del USART, y a menos que restaure los indicadores de error, no será posible la verificación de los datos que llegan.

Bit 3:SBRK - Poniendo un uno en este bit, forzamos a cero a la patilla 19.

Bit 2:RXE Receptor activado (Receptor enable) - Poniendo un uno en este bit activamos la parte de recepción del USART.

Bit 1:DTR - Poniendo este bit a uno, forzamos a cero la patilla 24, DTR.

Bit 0:TX EN Trasmisión activada (Transmit Enable) - Poniendo a uno este bit, activamos la sección de trasmisión del USART.

Cuando se inicializa el USART después del encendido, sería deseable, aunque no esencial, que se realizase una inicialización completa. Con esto nos aseguraríamos una restauración total, y que se programaría correctamente el modo de operación. Para ello debemos mandar al registro de control los siguientes seis octetos Hex en sucesión:

```
Octeto 1: 00  - Tres caracteres ASCII NUL.
Octeto 2: 00
Octeto 3: 00
Octeto 4: 40  - Programa una restauración interna.
Octeto 5: CE  - (o el octeto de modo que necesite) Este selecciona 2
                bits de parada: Sin paridad: 8 bits: Dividir por 16
Octeto 6: 15  - Restaura los indicadores de error y activa las sec-
                ciones de trasmisión (TX) y de recepción (RX).
```

Si quiere hacer una inicialización mínima del USART, elimine los tres primeros octetos y comience con el cuarto.

Cuando haya construido la tarjeta serie necesitará comprobar su correcto funcionamiento. El mejor método es conectar la línea de trasmisión de cada canal construido con la de recepción. Esto nos permitirá mandar caracteres y asegurarnos de que los podemos recibir correctamente. Así nos aseguramos no solamente del funcionamiento correcto del USART y su electrónica asociada, sino también de que el USART y el microprocesador del CPC se está comunicando correctamente.

La figura 3.51 es el listado de un programa para hacer una larga prueba de este tipo. Cuando haya conectado las partes de trasmisión y recepción del canal a probar, deberá ejecutar este programa. Nos mostrará el número de veces que se han encontrado puestos los indicadores de error del USART. El programa muestra también cuantos errores de datos han ocurrido, que es el número de veces que un octeto recibido es diferente del mandado. Para asegurarse del correcto funcionamiento del canal serie, el programa se debe ejecutar durante 30 minutos o más.

```
100  ' ** Este es un programa para probar el canal seleccionado de la     **
       ** tarjeta de E/S serie. Primero debe seleccionar el canal que se  **
       ** va a probar, 0-3, después inicializarlo y probarlo.             **
110  ' ** NOTA: debe conectar el canal a sí mismo. EJ conectando el cable **
       ** de trasmisión al de recepción.                                  **
       ** El programa manda al bucle todos los caracteres posibles, y     **
120  ' ** comprueba si se reciben correctamente. Los contadores de error  **
       ** en la pantalla muestran cuantos errores de cada tipo posible    **
       ** han ocurrido. Los tipos de error son, error de Paridad, error   **
```

```
130   ' ** de Estructura, error de Sobrecarga, y error de Datos. Todos los **
        ** errores, menos el último, se detectan en la lógica del USART.   **
        ** El error de datos significa que el programa ha recibido un      **
140   ' ** carácter distinto del que ha mandado.                           **

150   DIM BD(16),EQUIV(16) ' la matriz BD contiene la representación textual
      de las velocidades de trasmisión disponibles. EQUIV contiene el valor
      que se pone en el registro de velocidad del USART seleccionado
160   MODE 2: ON BREAK GOSUB 470: PRINT "Este programa hace una prueba contínua
      de uno de los canales de E/S serie": PRINT "que hay en la tarjeta
      serie";STRING$(5,10)
170   USARTBASE=&F9E0: GOSUB 320:
      INPUT "Que canal desea probar? (0 a 3) <0> ";C:
180   IF C < 0 OR C > 3 THEN 170 ELSE PRINT: PRINT :
      USARTBASE=USARTBAS+(C*2)
190   INPUT "Seleccione velocidad de trasmisión (ENTER muestra una lista)";B
200   IF B <= 0 THEN GOSUB 330: GOTO 190
210   FOR I=1 TO 16: IF BD(I) <> B THEN 230 ELSE IF C < 2 THEN
      OUT &F9E8,EQUIV(I) ELSE OUT &F9EC,EQUIV(I) ' Pone el valor seleccionado
      de velocidad de transmsión en el registro de velocidad adecuado
220   GOTO 240
230   NEXT I: MES$="Velocidad desconocida": GOSUB 330: GOTO 190
240   CLS: FOR I=1 TO 3: OUT USARTBASE+1,0: NEXT I:
      OUT USARTBASE+1,&40: OUT USARTBASE+1,&FE:
      OUT USARTBASE+1,&15 ' hacer inicialización total del USART
250   ' ** Para hacer la inicialización completa del USART se deben mandar 3
        ** octetos a cero, seguido de una restauración interna y la palabra
        ** de modo. (ver el texto) Después restauramos los indicadores de
255   ' ** error y activamos TX y RX
260   GOSUB 360 ' Dibujamos la pantalla
270   T=TIME: PAR=0: OV=0: FR=0: DAT=0: PASS=0: EVERY 3000,1 GOSUB 450
      ' ** Recuerde, al comenzar la ejecución debe poner los contadores a
        ** cero e inicializar los minutos de ejecución
280   FOR I=0 TO 255: WHILE (INP(USARTBAS+1) AND &1)=0: WEND:
      OUT USARTBASE,I: WHILE (INP(USARTBASE+1) AND &2)=0: WEND:
      IF INP(USARTBASE) <> I THEN DAT=DAT+1: GOSUB 420: GOTO 310
290   VER = (INP(USARTBASE+1) AND &38): IF VER AND &8 <> 0 THEN
      PAR=PAR+1 ELSE IF VER AND &10 <> 0 THEN OV=OV+1 ELSE IF VER
      AND &20 <> 0 THEN FR=FR+1
300   GOSUB 420: IF (INP(USARTBASE+1) AND &38) <> 0 THEN
      OUT (USARTBASE+1),&15 ' Limpiar los indicadores de error que haya
310   NEXT I: PASS=PASS+1: GOTO 280
320   RESTORE: FOR I=1 TO 16: READ BD(I),EQUIV(I): NEXT: RETURN
330   '
      ** Subrutina para mostrar las velocidades de trasmisión disponibles **

340   CLS: LOCATE 10,1: PRINT "Las velocidades de trasmisión son las siguientes";
      STRING$(2,10): FOR I=1 TO 16: PRINT BD(I): NEXT: PRINT:
      IF MES<>"" THEN PRINT MES$+CHR$(7);"-";B: MES$=""
350   RETURN
360   LOCATE 20,1: PRINT "Prueba del canal serie de E/S":
      FOR I=39 TO 339 STEP 300: MOVE I,100: DRAWR 230,0:
      DRAWR 0,230: DRAWR -230,0: DRAWR 0,-230: NEXT I
370   LOCATE 15,6: PRINT "Informe de errores": LOCATE 7,10:
      PRINT "Paridad=": LOCATE 7,12: PRINT "Sobrecarga=": LOCATE 7,14:
      PRINT "Estructura=": LOCATE 7,16: PRINT "Datos="
380   ' ** Los contadores de error están en la posición horizontal 27 en      **
        ** las líneas 10  (Paridad) 12 (Sobrecarga) 14  (Estructura) 16 (Datos) **
```

```
390   LOCATE 52,6: PRINT "Detalles de la prueba": LOCATE 45,8: PRINT
      "Canal seleccionado ";C: LOCATE 45,12: PRINT "Velocidad Trasmisión ";B:
      LOCATE 45,14: PRINT "Carácter mandado ="
400   LOCATE 45,16: PRINT "Pasadas =": LOCATE 45,18: PRINT
      "Tiempo de ejecución (mins)=": LOCATE 45,10: PRINT "Dirección canal E/S= ";
      HEX$(USARTBASE)
410   RETURN
420   '
      ** Actualización del estado en la pantalla **

430   LOCATE 20,10: PRINT PAR: LOCATE 20,12: PRINT OV: LOCATE 20,14:
      PRINT FR: LOCATE 20,16: PRINT DAT: LOCATE 63,14: PRINT I;"  ":
      LOCATE 63,16: PRINT PASS
440   RETURN
450   L=POS(#0): R=VPOS(#0): LOCATE 72,18: PRINT INT(((TIME-T)/300)/60):
      LOCATE L,R: RETURN '** Pequeña rutina para imprimir el tiempo CADA minuto
      - ver línea 270. Recuerda dónde estaba el cursor y lo vuelve a poner **
460   DATA 19200,0,19200,1,50,2,75,3,134.5,4,200,5,600,6,2400,7,
           9600,8,4800,9,1800,&A,1200,&B,2400,&C,300,&D,150,&E,
           110,&F,-1
470   LOCATE 6,22: R=REMAIN (1): PRINT
      "Programa abortado por ** BREAK **": END
```

Fig 3.51 Programa BASIC de prueba del canal RS232 VB

Ahora que ya hemos visto el funcionamiento y prueba del canal serie en el interfase RS232 versión B, veamos como ponerlo a trabajar.

El canal serie en funcionamiento

Consideremos un caso en el que usted quiere usar uno solo de los cuatro canales, conectado a un MODEM a 300 baudios. Este sería el caso si quisiera conectarse a una red de datos. El programa BASIC, que listamos en la figura 3.52, hará que el canal cero funcione a 300 baudios. En este caso se asume que solo quiere usar el canal a 300 baudios, y que el canal en uso será siempre el cero. No se tiene en cuenta la actividad de los otros canales - si están conectados. Como se puede ver en el listado, éste es un programa que pone simplemente en la pantalla todo lo que le llega, sin comprobar los errores ni asegurarse de que lo que le llega es un carácter imprimible. Esto significa que si pone mal la velocidad de trasmisión mandará a la pantalla del CPC gran cantidad de basura, incluyendo algunos caracteres de control, con resultados extraños en ciertas ocasiones.

```
100   '** Este programa convierte al CPC en una unidad de visualización serie
       ** conectado a una línea de comunicaciones de 300 baudios. Para ejecutarlo
       ** a velocidades superiores debe usar programas más complejos.
200   USARTBASE=&F9E0: CLS: FOR I=1 TO 3: OUT USARTBASE+1,0: NEXT I:
      MODE 2: OUT USARTBASE+1,&40: OUT USARTBASE+1,&CE: OUT &F9E4,&D:
      OUT USARTBASE+1,&15 ' Inicialización completa del USART a 300 baudios
300   '** Poner el USART 1 para que divida la frecuencia de su reloj por 16, para
       ** acomodarse a la salida de los chips 4702 que es 16 veces la velocidad
400   D$=INKEY$: IF D$="" THEN 600 ELSE OUT &F9E0,(ASC(D$))
500   '** La línea 200 mira si se ha pulsado una tecla, si no, salta a la
       ** línea 300. En caso afirmativo manda el carácter.
600   IF ((INP(&F9E1)) AND &2) =0 THEN 400 ELSE R=(INP(&F9E0) AND &7F):
      PRINT CHR$(R);: GOTO 400
```

```
700  END ' ** La línea 300 mira si se ha recibido el carácter en el USART 1.
     ** Si se ha recibido lo toma. NOTA: En esta versión se asume que el carácter
     ** recibido es correcto, no se analizan los señalizadores de error.
```

Fig 3.52 Programa BASIC mínimo para manejar un canal serie

Este programa será adecuado para muchas aplicaciones de los canales de E/S serie, pero tiene serias limitaciones para usos que requieran velocidades de trasmisión mayores de 300 baudios, más flexibilidad, u operaciones multi-canal. El programa de manejo del canal serie que se describe en la próxima sección nos proporcionará todas estas funciones.

Programa de manejo del canal serie

Veamos como podemos manejar el 'hardware' como una VDU multicanal.

El programa que vamos a presentarle fue escrito originalmente en BASIC. Por desgracia, aunque el Locomotive BASIC es uno de los intérpretes BASIC más rápidos, el programa resultante no podía manejar velocidades superiores a 1200 baudios. Por lo tanto hemos tenido que reescribir el programa, con las rutinas más críticas en tiempo escritas en ensamblador para generarlas en código máquina. No se preocupe si no entiende muy bien los listados de ensamblador de las páginas siguientes, podrá volver a ellas después de leer el capítulo 5, que es una introducción a la programación en lenguaje ensamblador.

El programa de manejo del canal serie proporciona las siguientes características:

1) Operación tipo VDU en un canal seleccionado.
2) Recepción de mensajes limitada en los canales no seleccionados
3) Operación hasta a 19200 baudios
4) Un modo especial donde se pueden poner las siguientes características para cualquier canal:

   A) Velocidad de trasmisión
   B) Longitud de palabra
   C) Numero de bits de parada
   D) Paridad puesta/quitada y selección par/impar

El menú de preparación muestra adicionalmente los valores actuales e indica cuándo hay mensajes pendientes en los canales no seleccionados.

Como se prevé que el programa sea usado con diferentes sistemas, no se realiza interpretación de códigos de control, esto no es un paquete de emulación para un tipo particular de VDU - pienso que no es difícil la adaptación para un tipo en particular. Ya que no responde a los códigos especializados, el sistema debe tratar al CPC como un terminal mudo.

El controlador serie aparece listado en la figura 3.53. Está realizado en BASIC con el código máquina incluido en sentencias DATA. (La técnica para añadir el código máquina a un programa BASIC se trata en el capítulo 5). Las instrucciones de manejo del programa son muy simples. Primero debe cargar el programa desde el disco o la cinta, después debe ejecutarlo. Comienza con una pantalla en blanco en la que

solamente aparece el cursor - como en la mayoría de las VDU disponibles comercialmente. Si no desea cambiar las opciones por defecto (listadas más adelante) debe establecer la conexión con el otro ordenador y usar el CPC como una VDU normal.

Para que aparezca el menú de preparación, debe pulsar CTRL/P o la tecla CLR situada sobre la tecla grande de ENTER.

Si introduce el programa, tal como se ve en la figura 3.53, las opciones por defecto cuando lo ejecute por primera vez serán:

```
Velocidad=9600     Bits de parada=2          Paridad=No
Dividir por=16     Longitud de palabra=8     Caracteres pendientes=NO
```

Ya que no podemos mostrar más de una pantalla completa a la vez (a menos que dividamos la pantalla en cuatro ventanas más pequeñas) necesitamos asegurarnos que solo veremos los datos que vienen de un canal. A este canal le llamaremos el canal seleccionado, y se puede especificar como cualquiera de los canales entre cero y tres en el menú de selección. En la pantalla del menú se marcan las opciones seleccionadas.

```
500  '**    Controlador serie versión 4F:  Abril 1985

1000 '** Este es el controlador del interface serie de 4 canales. Está escrito en
una
      ** mezcla de BASIC y ensamblador.
4000 '** Para obtener más detalles del uso de la memoria, lea el texto del libro,
      ** pero se resume en lo siguiente: HIMEM del BASIC es Hex 57FF
      ** Las memorias intermedias están situadas desde Hex 6000 a Hex 63FF
5000 '** USARTINIT se carga en Hex 6440: Hay algunas posiciones específicas de
      ** memoria que se usan para la intercomunicación entre el BASIC y el código
      ** máquina. El programa no co-residirá en memoria.
7000 '**                    Comienzo

7005 MEMORY &57FF: MODE 1: LOCATE 10,5: DIM BAUD$(16): BORDER 5,10:
     SPEED KEY 15,1: PRINT "Inicializando E/S serie. Por favor espere":
     PRINT
7010 WHILE R$ <> "USARTINIT": READ R$: WEND: FOR M=&6440 TO &64C0:
     READ TMP: POKE M,TMP: NEXT M: PRINT "USARTINIT  instalado"
     '** Introduce USARTINIT en memoria en Hex 6440
7030 BORDER 26: RESTORE: WHILE R$<> "MACCODE": READ R$: WEND:
     FOR M=&6500 TO &65C0: READ TMP: POKE T,TMP: NEXT:
     PRINT "Código máquina instalado": INK 2,24,10: SPEED INK 10,20
8010 RESTORE: FOR T=&6400 TO &6413: READ TMP: POKE T,TMP:
     NEXT '** Esto trasfiere los valores por defecto en las sentencias DATA
     ** a la memoria intermedia de parámetros, usados por la rutina USARTINIT
8020 CALL &6440: PRINT "Instaladas opciones por defecto": PRINT: PEN #0,2: PRINT
     "Pulse CTRL/P o CLR para modificar opciones": PEN #0,1
8030 FOR I=26 TO 1 STEP -1: FOR P=1 TO 75: BORDER 1: NEXT: NEXT
     '** Para que haga bonito mientras el usuario lee la información
8040 POKE &6003,&FF: POKE &6103,&FF: POKE &6203,&FF: POKE &6303,&FF
     '** Inicializa las memorias intermedias de datos del canal, con EOBs
8060 SELCHAN=0: POKE &6414,0 ' ** Selecciona primero el canal 0
8070 MODE 2: CALL &6500 '** Ir al código KERNEL
10000 '** Esta rutina imprime las opciones del canal seleccionado, también
      ** imprime los contadores de error de todos los canales. Permite
      ** seleccionar un nuevo canal e indica si se ha recibido algún dato
```

```
10010  ** no visualizado en algún canal no seleccionado.

10015 OLDSEL=SELCHAN: R$=CHR$(19): FOR I=0 TO 3: SELCHAN=I:
      GOSUB 11020: NEXT I: SELCHAN=OLDSEL
      '** Manda un XOFF a cada canal mientras estamos en el menú
10020 BORDER 5: CLG: MOVE 1,1: FOR I=100 TO 300 STEP 100: MOVE 1,I:
      DRAWR 600,1: MOVER 0,1: DRAWR -600,1: NEXT: FOR I=0 TO 3:
      LOCATE 1,2+(I*6)
10030 PRINT CHR$(24);" Canal";I;CHAR$(24);
      "      ERRORES: Parid. =      : Estruc.=      : Sobrec.=      ":
      IF I=SELCHAN THEN PRINT: PRINT "* Seleccionado *"
10040 LOCATE 17,4+(I*6): PRINT
      "OPCIONES:  Veloc=       : Bits=       : Parid.=": PRINT TAB(18);
      "      Bits par.=       : Caracteres recibidos pend= "
10045 NEXT
10050 '** Ahora que hemos impreso los títulos de todos los datos, tenemos
       ** que imprimir los valores
10055 IF BAUD$(0)="" THEN GOSUB 42000
10060 FOR I=0 TO 3: LOCATE 35,(I*6)+2: PRINT PEEK(&6000+(I*&100)):
      LOCATE 49,(I*6)+2: PRINT PEEK(&6000+(I*&100)):
      LOCATE 65,(I*6)+2: PRINT PEEK(&6000+(I*&100))
10070 '** La línea anterior imprime los contadores de error
10080 LOCATE 35,(I*6)+4: PRINT BAUD$(PEEK(&6400+(I*5))):
      LOCATE 47,(I*6)+4: PRINT 5+(PEEK(&6402+(I*5))):
      LOCATE 65,(I*6)+4
10090 IF (PEEK(&6403+(I*5)) AND 1)=0 THEN PRINT "No" ELSE
      PRINT "Si-  ";: IF (PEEK(&6403+(I*5)) AND 2)<>0 THEN PRINT
      "PAR " ELSE PRINT "IMPAR"
10100 LOCATE 34,(I*6)+5: R=(PEEK(&6404+(I*5))): IF R=3 THEN PRINT
      2 ELSE PRINT R
10105 LOCATE 72,(I*6)+5: IF PEEK(&6003+(I*&100)) =255 THEN PRINT
      "NO" ELSE PRINT "SI"
10110 NEXT: BORDER 1: LOCATE 1,25: R$="": PRINT CHR$(24);
      " Pulse R para volver a VDU o S para cambiar las opciones";
      CHR$(24);: WHILE R$ <> "R" AND R$ <> "S"
10120 R$=UPPER$(INKEY$): WEND: IF R$="R" THEN GOTO 10500
      '** Si ha elegido R, vuelve al VDU: en caso contrario ir
       ** al menú de cambios
10125 '**      Esta sección permite al usuario cambiar las opciones      **
10130 CLS: TITLE$="Cambio opciones del canal serie": GOSUB 41000
10140 LOCATE 4,5: MES$="El canal seleccionado actualmente es":
      PRINT CHR$(20);MES$; SELCHAN: LOCATE 4,7:
      INPUT "Numero de canal a seleccionar - ENTER si no hay cambios";R$:
10150 IF R$="" THEN GOTO 10160 ELSE IF VAL(R$) > 3 OR VAL(R$) < 0 THEN
      GOTO 10140 ELSE SELCHAN=INT(VAL(R$)): POKE &6414,SELCHAN:
      LOCATE 4,5: PRINT CHR$(20);MES$;SELCHAN '** Nueva tabla de selección
      de canal en memoria
10160 LOCATE 4,7: BASE=(&6400+(SELCHAN*5)): PRINT "Paridad ";CHR$(20);:
      IF PEEK(BASE+3) AND 1)=0 THEN PRINT "<No>"; ELSE
      PRINT "<Si>";
10170 INPUT R$: IF UPPER$(LEFT$(R$,1))="S" OR R$="" THEN 10180
      ELSE POKE BASE+3,0: GOTO 10190
10180 LOCATE 4,7: PRINT CHR$(20);: IF (PEEK(BASE+3) AND 1)=0 AND UPPER$
      (LEFT$(R$,1)) <> "S" THEN 10190 ELSE INPUT "Par O Impar ";R$:
      IF UPPER$(LEFT$(R$,1))="P" THEN POKE BASE+3,3 ELSE POKE BASE+3,1
10190 LOCATE 4,7: PRINT CHR$(20);"Número de bits por carácter <";
      (5+(PEEK(BASE+2)));">";: INPUT R$: R=VAL(R$): IF R=0 THEN
      R=(5+(PEEK(BASE+2))) ELSE IF R >8 OR R<5 THEN 10190
10200 POKE BASE+2,R-5
```

```
10210 LOCATE 4,7: PRINT CHR$(20);"Velocidad <";BAUD$(PEEK(BASE));">";:
      INPUT R$: IF R$="" THEN GOTO 10230 ELSE FOR I=0 TO 15:
      IF R$=BAUD$(I) THEN 10220 ELSE NEXT I: GOTO 10210
10220 IF PEEK(&6414) > 1 THEN 10225 ELSE POKE &6400,I: POKE &6405,I:
      GOTO 10230
10225 POKE &640A,I: POKE &640F,I
10230 LOCATE 4,7: PRINT CHR$(20);"Número de bits de parada: 1 o 2 <2>";:
      INPUT R$: IF R$="" THEN 10240 ELSE IF VAL(R$) <=0 THEN 10230 ELSE
      IF VAL(R$) > 2 THEN 10230
10235 IF INT(VAL(R$))=1 THEN POKE BASE+4,1 ELSE IF INT(VAL(R$))=2
      THEN POKE BASE+4,3 ELSE 10230
10240 CALL &6440: GOTO 10000 '** Con esto terminamos las opciones. Para hacerlo
      más simple solo permitimos 1 o 2 bits de parada. Ahora instalaremos los
      cambios y rearrancaremos la rutina de pantalla
10500 CLS: SELCHAN=OLDSEL: R$=CHR$(17): FOR I=0 TO 3: SELCHAN=I:
      GOSUB 11020: NEXT I: SELCHAN=OLDSEL: GOTO 8070
      '** Decir a todos que volvemos al XON y terminar.
11000 '** Rutina para mandar el caracter tecleado en el teclado del CPC
       ** al canal seleccionado. Si el caracter tecleado es = 16, ejecutaremos
       ** la subrutina del menú. En 'HALF DUP' lo sacamos tambien a pantalla.
11020 WHILE INP(&F8E10+(SELCHAN*2)) AND &2 = 0: WEND:
      OUT (&F8E0+(SELCHAN*2)),ASC(R$)
11090 RETURN
20000 '** Las siguientes sentencias DATA representan los parámetros para
       ** cada uno de los 4 canales serie. Estos parámetros están totalmente
       ** definidos en el texto. Los valores que aparecen en esta lista
20010  ** ponen estos parámetros con los siguientes valores por defecto:
       ** 9600 baudios : Dividir las frecuencias de reloj de RX y TX por 16:
       ** trasferir 8 bits: Sin paridad: 2 bits de parada. La sentencia DATA
20020  ** 20030 pone las opciones del canal 1: la 20040 las del canal 2, y así
       ** sucesivamente...

20030 DATA 8,2,3,0,3
20040 DATA 8,2,3,0,3
20050 DATA 8,2,3,0,3
20060 DATA 8,2,3,0,3
20100 DATA "VELOCIDAD DE TRASMISION"
20110 DATA 19200,19200,50,75,134.5,200,600,2400,9600,4800,1800,
          1200,2400,300,150,110
30999 DATA "USARTINIT"
31000 DATA 221 , 229 , 213 , 229 , 245 , 197 , 46 , 0 , 62 , 3
31010 DATA 205 , 176 , 100 , 46 , 64 , 62 , 1 , 205 , 176 , 100
31020 DATA 221 , 33 , 0 , 100 , 17 , 5 , 0 , 1 , 232 , 249
31030 DATA 221 , 126 , 0 , 237 , 121 , 1 , 236 , 249 , 221 , 126
31040 DATA 10 , 237 , 121 , 1 , 225 , 249 , 46 , 4 , 221 , 126
31050 DATA 1 , 221 , 102 , 2 , 203 , 4 , 203 , 4 , 180 , 221
31060 DATA 102 , 3 , 203 , 4 , 230 , 4 , 203 , 4 , 203 , 4
31070 DATA 180 , 221 , 102 , 4 , 203 , 4 , 203 , 4 , 203 , 4
31080 DATA 203 , 4 , 203 , 4 , 203 , 4 , 180 , 237 , 121 , 45
31090 DATA 40 , 6 , 3 , 3 , 221 , 25 , 24 , 206 , 46 , 21
31100 DATA 62 , 1 , 205 , 176 , 100 , 193 , 241 , 225 , 209 , 221
31110 DATA 225 , 201 , 30 , 4 , 1 , 225 , 249 , 87 , 237 , 105
31120 DATA 21 , 32 , 251 , 29 , 200 , 3 , 3 , 24 , 244 , 0
31999 DATA "MACCODE"
32000 DATA 245 , 213 , 197 , 229 , 205 , 129 , 187 , 205 , 98 , 101
32010 DATA 58 , 20 , 100 , 246 , 96 , 103 , 46 , 3 , 203 , 126
32020 DATA 32 , 9 , 126 , 205 , 90 , 187 , 205 , 72 , 101 , 24
32030 DATA 232 , 205 , 9 , 187 , 48 , 227 , 254 , 16 , 32 , 10
32040 DATA 205 , 132 , 187 , 225 , 193 , 209 , 241 , 201 , 24 , 206
```

```
32050 DATA 95 , 1 , 225 , 249 , 58 , 20 , 100 , 203 , 39 , 129
32060 DATA 79 , 237 , 120 , 230 , 1 , 40 , 250 , 13 , 237 , 89
32070 DATA 24 , 191 , 229 , 245 , 58 , 20 , 100 , 230 , 3 , 246
32080 DATA 96 , 103 , 46 , 3 , 203 , 126 , 32 , 7 , 35 , 126
32090 DATA 43 , 119 , 35 , 24 , 245 , 225 , 241 , 201 , 245 , 197
32100 DATA 229 , 213 , 1 , 225 , 249 , 33 , 0 , 96 , 22 , 4
32110 DATA 237 , 120 , 11 , 254 , 255 , 40 , 36 , 203 , 79 , 40
32120 DATA 32 , 230 , 56 , 32 , 48 , 125 , 198 , 3 , 111 , 203
32130 DATA 126 , 32 , 3 , 35 , 24 , 249 , 237 , 120 , 230 , 127
32140 DATA 119 , 35 , 54 , 255 , 62 , 255 , 189 , 32 , 4 , 46
32150 DATA 3 , 54 , 255 , 21 , 32 , 5 , 209 , 225 , 193 , 241
32160 DATA 201 , 46 , 0 , 124 , 198 , 1 , 103 , 121 , 198 , 3
32170 DATA 79 , 24 , 193 , 203 , 95 , 40 , 1 , 52 , 35 , 203
32180 DATA 103 , 40 , 1 , 52 , 35 , 203 , 111 , 52 , 40 , 1
32190 DATA 52 , 62 , 21 , 3 , 237 , 121 , 11 , 24 , 210 , 0
32200 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
41000 '** Rutina para imprimir TITLE$ en un recuadro en la línea superior
41010 SYMBOL 255,&E0,&E0,&E0,&E0,&E0,&E0,&E0,&E0: SYMBOL 254,&7,&7,&7,&7,&7,
      &7,&7,&7
41020 TITLE$=CHR$(255)+" "+TITLE$+" "+CHR$(254)
41030 LEFT%=(40-(LEN(TITLE$)/2)): LOCATE LEFT%,2: PRINT TITLE$
41040 MOVE (8*(LEFT%-1)),384
41050 DRAW 8*(LEFT%+ (LEN(TITLE$)-1))-1,384
41060 MOVE XPOS,(YPOS-18): DRAW 8*(LEFT%-1),YPOS
41070 RETURN
42000 '** Esta rutina prepara la matriz BAUD$ que contiene la
      ** representación real de las velocidades de trasmisión
42010 RESTORE: WHILE R$ <> "VELOCIDAD DE TRASMISION": READ R$: WEND:
      FOR I=0 TO 15: READ BAUD$(I): NEXT: RETURN
```

Fig 3.53 Controlador del interface de cuatro canales

Los tres canales no seleccionados pueden recibir aún hasta 253 caracteres de cualquier sistema o dispositivo al que estén conectados y, cuando uno de estos canales pase a ser el canal seleccionado, se mostrarán los mensajes pendientes. El menú de cambios indicará también si hay caracteres pendientes para un canal.

Las modificaciones que se hagan en el menú de cambios se implementan al salir de la pantalla del menú.

Hagamos ahora un repaso del programa:

El programa usa posiciones fijas de memoria como memorias intermedias, y para mantener los módulos en código máquina. Estas son:

```
HEX 6000-60FF = Memoria de recepción de datos del canal 0
HEX 6100-61FF = Memoria de recepción de datos del canal 1
HEX 6200-62FF = Memoria de recepción de datos del canal 2
HEX 6300-63FF = Memoria de recepción de datos del canal 3
HEX 6414      = Número del canal seleccionado
HEX 6440-64C0 = USARTINIT módulo en código máquina
HEX 6500-65C0 = MACCODE módulo en código máquina
```

Cada canal tiene una memoria intermedia de datos recibidos, cuya longitud es 256 octetos. Solamente se usan 253 octetos para almacenar caracteres, y los tres primeros octetos se usan como contadores de errores del canal. Estos están dispuestos de la siguiente forma:

Canal 0 dirección Hex 6000)
Octeto 1 = Contador de errores de paridad.
Octeto 2 = Contador de errores de estructura
Octeto 3 + Contador de errores de sobrecarga

El octavo octeto es el principio de la memoria intermedia. Ya que este programa está diseñado para manejar solamente códigos ASCII - que usan solamente los siete bits inferiores de cada octeto - podemos usar el bit más alto como señalizador para que el gestor de la memoria intermedia sepa dónde está el final de la misma. Por lo tanto, si un octeto de la sección de datos tiene el bit siete puesto, significa que es el último carácter en la memoria intermedia.

NOTA: Si, mientras el canal está deseleccionado, recibe más de 253 caracteres, el carácter 254 es colocado en el primero octeto de la memoria intermedia, y el contenido anterior se pierde. Esto se hace así porque el dispositivo de recepción solo está diseñado para la recepción de mensajes, no para la recepción normal de datos.

La memoria intermedia de parámetros contiene los parámetros por defecto, que usa el programa en la definición del octeto de modo de cada USART durante la ejecución de la rutina USARTINIT. La memoria intermedia de parámetros está situada en las posiciones de memoria Hex 6400 a Hex 6413. Hay cinco parámetros asociados con cada canal, que se ponen desde los valores de los parámetros que hay en la memoria intermedia. Estos son:

1) Valor del registro de velocidad de trasmisión del canal
2) Número de bits de parada
3) Factor de división para el USART
4) Longitud de palabra
5) Paridad
6) Número de bits de parada

Todos estos valores, excepto el de velocidad de trasmisión, los usa la rutina USARTINIT para construir el octeto de modo que se colocará en el USART, después de restaurarlo. Referirse a la figura 3.48 para obtener más detalles del octeto de modo. Veamos este área con más detalle. La línea 20030 del programa es como sigue:

```
20030 DATA 8,2,3,0,3
```

Con estos valores ponemos los parámetros por defecto del canal 0. El ocho es el valor colocado en el registro de selección de velocidad de trasmisión para seleccionar 9600 baudios. El dos se usa en USARTINIT como los dos bits inferiores de la palabra de modo para seleccionar la división por dieciséis. El primer tres selecciona 8 bits de datos. El cero inactiva la comprobación de paridad. El segundo tres selecciona 2 bits de parada. Si nos referimos a la figura 3.48 veremos este esquema más claramente.

La línea 20030 pone los parámetros del canal cero, la línea 20040 pone los del canal uno, y así sucesivamente.

Después de instalar el programa en código máquina y poner los parámetros por defecto, la sección del BASIC llama a USARTINIT para inicializar los USART. En la figura 3.54 damos el listado en ensamblador del programa USARTINIT. Una vez más, los lectores que no estén familiarizados con el ensamblador pueden saltarse el listado y volver a él cuando hayan leído el capítulo 5.

```
Pass 1 errors: 00

                100 ;      ** PROGRAMA DE INICIALIZACION DEL USART
                200 ;
                300 ; ** Este programa es para el interfase RS232 de cuatro
                400 ; ** canales, cuyos detalles se dan en el libro
                500 ;
                600 ; ** Este programa inicializa los 4 canales - y usa los
                700 ; ** parámetros del controlador serie en BASIC
                800 ;
                900 ;
6440            1000        ORG  #6440      ; Define dónde estamos ejecutando
6440            1100 INITOS ENT  $
6440  DDE5      1200        PUSH IX
6442  D5        1300        PUSH DE
6443  E5        1400        PUSH HL
6444  F5        1500        PUSH AF
6445  C5        1600        PUSH BC     ; Preparar espacio
6446  2E00      1700        LD   L,#0   ; L es el caracter a mandar
6448  3E03      1800        LD   A,#3   ; A indica las veces que hay que mandarlo
644A  CDB064    1900        CALL SENDIT     ; Llama a la rutina de mandar
644D  2E40      2000        LD   L,#40 ; Ahora manda Hex 40 una vez
644F  3E01      2100        LD   A,#1   ; para restaurar el USART
6451  CDB064    2200        CALL SENDIT     ; y manda el comando de restaurar
                2300 ;
                2400 ; Ahora que tenemos cada USART esperando el octeto de
                2500 ; modo (que se describe en el texto), nos preparamos para
                2600 ; poner los parámetros de cada canal - que ha colocado en
                2700 ; una pequeña memoria intermedia que comienza en la
                2800 ; dirección de memoria Hex 6400, el programa BASIC principal
                2900 ; - en cada USART
                3000 ;
6454  DD210064  3100 MODSET LD   IX,#6400   ; IX es base de los parámetros
6458  110500    3200        LD   DE,#5      ; DE=valor actualizado a sumar a IX
645B  01E8F9    3300        LD   BC,#F9E8   ; Carga reg velocidad canales 0 y 1
645E  DD7E00    3400        LD   A,(IX)     ; Carga valor velocidad
6461  ED79      3500        OUT  (C),A      ; Lo pone en el registro
6463  01ECF9    3600        LD   BC,#F9EC   ; Carga reg velocidad canales 2 y 3
6466  DD7E0A    3700        LD   A,(IX+10)  ; Carga valor velocidad
6469  ED79      3800        OUT  (C),A      ; Lo pone en el registro
646B  01E1F9    3900 SETUP  LD   BC,#F9E1   ; Ahora el resto de los parámetros
646E  2E04      4000        LD   L,4        ; Prepara cuatro bucles
6470  DD7E01    4100 SETLOP LD   A,(IX+1)   ; A=factor división para este canal
6473  DD6602    4200        LD   H,(IX+2)   ; H=num de bits que hay que mandar
6476  CB04      4300        RLC  H          ; mover 2 bits a la izquierda
6478  CB04      4400        RLC  H
647A  B4        4500        OR   H          ; poner el resultado en A
647B  DD6603    4600        LD   H,(IX+3)   ; Poner paridad y par/impar
647E  CB04      4700        RLC  H          ; mover 4 bits a la izquierda
6480  CB04      4800        RLC  H
6482  CB04      4900        RLC  H
6484  CB04      5000        RLC  H
6486  B4        5100        OR   H          ; Poner selección paridad en A
6487  DD6604    5200        LD   H,(IX+4)   ; Poner los bits de parada
648A  CB04      5300        RLC  H          ; mover 6 bits a la izquierda
648C  CB04      5400        RLC  H
648E  CB04      5500        RLC  H
6490  CB04      5600        RLC  H
6492  CB04      5700        RLC  H
```

```
6494  CB04      5800        RLC  H          ; Y una vez hecho ponerlo dentro
6496  B4        5900        OR   H          ; de A. A = modo, pasarlo al USART
6497  ED79      6000        OUT  (C),A      ; seleccionado
6499  2D        6100        DEC  L          ; Decrementar L y saltar si se han
649A  2806      6200        JR   Z,ENALL    ; completado todos los USART
649C  03        6300        INC  BC         ; BC apunta al registro del
649D  03        6400        INC  BC         ; siguiente USART
649E  DD19      6500        ADD  IX,DE      ; Mover IX a la siguiente entrada
64A0  18CE      6600        JR   SETLOP     ; Y repetir el bucle
                6700 ;
                6800 ;    Ya hemos inicializado todos los USART. Ahora necesitamos
                6900 ; activarlos a todos. Lo hacemos usando SENDIT de nuevo.
64A2  2E15      7000 ENALL  LD   L,#15       ; Mandamos Hex 15 al USART
64A4  3E01      7100        LD   A,#1        ; y solo lo mandamos una vez
64A6  CDB064    7200        CALL SENDIT      ; Llama a SENDIT
                7300 ;
64A9  C1        7400        POP  BC
64AA  F1        7500        POP  AF
64AB  E1        7600        POP  HL
64AC  D1        7700        POP  DE
64AD  DDE1      7800        POP  IX          ; Restaurar los registros de entrada
                7900 ; Ahora tenemos los USART preparados para su uso
64AF  C9        8000        RET
                8100
                8200 ; Fin de USARTINIT.
                8300
                8400
                8500 ; Esta es la rutina SENDIT. Manda el carácter que recibe
                8600 ; en el registro L, a los registros de control de los 4 USART
                8700 ; n veces - donde n es el valor de A. También usa DE.
                8800 ; A la salida los registros A,BC,D,E son inválidos.
64B0  1E04      8900 SENDIT LD   E,#4        ; Preparar para 4 USART
64B2  01E1F9    9000        LD   BC,#F9E1    ; BC=1er registro estado USART
64B5  57        9100 SAV000 LD   D,A         ; Salvar el contador en D
64B6  ED69      9200 SAV001 OUT  (C),L       ; mandar el carácter
64B8  15        9300        DEC  D           ; Decrementar contador bucle
64B9  20FB      9400        JR   NZ,SAV001   ; Repetir si no ha terminado
64BB  1D        9500        DEC  E           ; ver si se han terminado los USART
64BC  C8        9600        RET  Z           ; retornar si se han terminado
64BD  03        9700        INC  BC          ; si no, BC apunta al siguiente
64BE  03        9800        INC  BC          ; registro de control de USART
64BF  18F4      9900        JR   SAV000      ; repetir de nuevo

Pass 2 errors: 00

ENALL  64A2   INITOS 6440   MODSET 6454   SAV000 64B5
SAV001 64B6   SENDIT 64B0   SETLOP 6470   SETUP  646B


Table used:    115  form    522
Executes: 25664
```

Fig 3.54 Listado ensamblador de USARTINIT

Cuando se ha completado USARTINIT, devuelve el control a la sección BASIC, que llama inmediatamente a la rutina principal en código máquina MACCODE. MACCODE consiste realmente en cuatro rutinas. Estas se

llaman KERNEL, INCOME, SEND y SHUFFLE. Veremos estas cuatro rutinas más detalladamente dentro de un momento.

En términos generales, el módulo principal del programa examina continuamente los chips USART instalados, para ver si han recibido algún carácter. Si lo han recibido, comprueba los señalizadores de error, actualizando los contadores de error de ese canal si es necesario. Si ha recibido los caracteres correctamente, los coloca en el siguiente octeto disponible en la memoria intermedia de recepción de ese canal. La otra función que realiza continuamente el programa es llamar a la ROM del CPC para ver si se ha tecleado algún carácter en el teclado. Si se ha tecleado alguno, se manda al USART del canal seleccionado, asumiendo que está dispuesto para trasmitir un carácter. La única excepción es si el carácter tecleado corresponde a CTRL/P o CLR (que generan el carácter 16), en cuyo caso se efectúa un retorno al BASIC.

Cuando la sección de código máquina devuelve control al BASIC, ésta pasa control a la rutina del menú de opciones. Esta rutina imprime las opciones de todos los canales, y usted debe teclear "R" para volver a la sección de código máquina y seguir usando su CPC como una VDU. También puede pulsar "S" para pasar a las preguntas que cambian las opciones. Estas son autoexplicativas, todos los valores introducidos como nuevos parámetros de canal se comprueban para evitar poner valores inválidos. Después de pasar por la sesión de preguntas y respuestas, la sección de BASIC introduce los nuevos valores dentro de la memoria intermedia de parámetros, y después llama a USARTINIT para implementar los cambios que se hayan hecho. Luego vuelve a mostrar el menú de nuevo. Durante el tiempo en que el programa está en el menú, se manda un carácter XOFF a todos los canales para que no manden caracteres que se podrían perder. (Vea la sección 3.2 - proyecto uno - para una descripción del significado de XON y XOFF).

Hay otro par de rutinas en la sección de BASIC del programa. Estas rutinas realizan la construcción de la pantalla o dan información. Las notas del listado explican su función.

```
Hisoft GENA3 Assembler.

Pass 1 errors: 00

                100 ;
                200 ;
                210 ; ** Módulo MACCODE controlador serie - Versión 4H
                211 ;
                300 ; ** La sección de código máquina del controlador del
                400 ; ** interfase serie de 4 canales. Se describe en el
                500 ; ** libro.
                600 ;
                700 ; ** Hay cuatro rutinas KERNEL, INCOME, SEND y SHUFFL.
                800 ; ** Cada una de estas rutinas está descrita dentro
                900 ; ** del texto del libro.
               1000
6500           1100        ORG  #6500      ; Empezamos detrás de USARTINIT
6500           1200        ENT  $
6500  F5       1300 KERNEL PUSH AF     ; Salva registros
6501  D5       1400        PUSH DE
6502  C5       1500        PUSH BC
```

```
6503 E5       1600        PUSH HL
6504 CD81BB   1700        CALL #BB81 ; Poner el cursor
6507 CD6265   1800 K1     CALL INCOME     ; Comprobar caracteres de entrada
650A 3A1464   1900        LD   A,(#6414)  ; Obtiene num canales seleccionados
650D F660     2000        OR   #60
650F 67       2100        LD   H,A        ; Conv. a dirección memoria en HL
6510 2E03     2200        LD   L,#03      ; Mem. Int empieza en 6x03 (texto)
6512 CB7E     2300        BIT 7,(HL)      ; EOB en principio mem. int.?
6514 2009     2400        JR   NZ,KEYCHK  ; si, saltar
6516 7E       2500        LD   A,(HL)     ; no, obtener e imprimir carácter
6517 CD5ABB   2600        CALL #BB5A
651A CD4865   2700        CALL SHUFFLE    ; Mueve la mem. int hacia arriba
651D 18E8     2800        JR   K1         ; y repite el bucle de nuevo.
651F CD09BB   2900 KEYCHK CALL #BB09      ; Llama a KM READ CHAR (ROM)
6522 30E3     3000        JR   NC,K1      ; Bucle si no se ha pulsado tecla
6524 FE10     3100        CP   #10        ; ver si se ha pulsado CTRL/P
6526 200A     3200        JR   NZ,SEND    ; no, mandar el carácter
6528 CD84BB   3300        CALL #BB84      ; quitar cursor
652B E1       3400        POP  HL
652C C1       3500        POP  BC
652D D1       3600        POP  DE
652E F1       3700        POP  AF         ; Volver al BASIC
652F C9       3800        RET             ; Así que, adios por ahora
6530 18CE     3900        JR   KERNEL     ; Con la opción de ejecutar de nuevo
              4000
              4100 ; *      Rutina de mandar
              4200 ; ** Mandar el carácter contenido en el registro A al
              4300 ; ** USART del canal seleccionado.
              4400 ;
6532 5F       4500 SEND   LD   E,A        ; Salvar el carácter tecleado
6533 01E1F9   4600        LD   BC,#F9E1   ; Cargar dirección prim USART
6536 3A1464   4700        LD   A,(#6414)  ; Obtener num canal seleccionado
6539 CB27     4800        SLA  A          ; Doblarlo
653B 81       4900        ADD  A,C        ; Añadirlo a octeto inf. dirección
              5000 ; Esto significa que si por ejemplo el canal seleccionado
              5100 ; fuera 1, después de doblarlo sería 2, y cuando sa haya
              5200 ; añadido a BC, este contendrá F9E3 - dirección del registro
              5300 ; de estado del canal 1 - ver texto.
653C 4F       5400        LD   C,A        ; Poner el valor final en C
653D ED78     5500 SEND01 IN   A,(C)
653F E601     5600        AND  #1    ; Tomar valor reg. estado - TX y RDY=1
6541 28FA     5700        JR   Z,SEND01   ; Saltar si no
6543 0D       5800        DEC  C          ; Decrementar C,apunta a reg DAT
6544 ED59     5900        OUT  (C),E      ; Mandar el carácter tecleado
6546 18BF     6000        JR   K1         ; y volver a KERNEL
              6100
              6200 ; **              SHUFFLE
              6300 ; ** Esta rutina mueve el contenido de la memoria intermedia
              6400 ; ** del canal serie, un octeto hacia arriba, esto es borra
              6500 ; ** el primer carácter y mueve el resto hacia arriba una
              6600 ; ** posición.
              6700 ; ** El número del canal se encuentra en la posición
              6800 ; ** absoluta de memoria Hex 6414.
              6900 ;
              7000 ; ** Este es un módulo del controlador de I/F serie
              7100
              7200
6548 E5       7300 SHUFFL PUSH HL
6549 F5       7400        PUSH AF
```

```
654A 3A1464    7500        LD   A,(#6414)  ; Obtiene el número del canal
654D E603      7600        AND  #3         ; A=numero de canal: Límite 0-3
654F F660      7700        OR   #60        ; y sumarle Hex 60
6551 67        7800        LD   H,A        ; Mult. por Hex 100, dir mem int
6552 2E03      7900        LD   L,03       ; L=inferior apuntador a dirección
6554 CB7E      8000 SHUF01 BIT  7,(HL)     ; ver si es EOB (bit siete = 1)
6556 2007      8100        JR   NZ,GOUT    ; Si lo es saltar
6558 23        8200        INC  HL
6559 7E        8300        LD   A,(HL)     ; Tomar siguiente octeto
655A 2B        8400        DEC  HL         ; Reajustar HL
655B 77        8500        LD   (HL),A     ; y moverlo uno hacia arriba
655C 23        8600        INC  HL         ; Incrementar apuntador
655D 18F5      8700        JR   SHUF01     ; y repetir
655F E1        8800 GOUT   POP  HL
6560 F1        8900        POP  AF
6561 C9        9000        RET
               9100
               9200
               9300
               9400 ;   * Esta es la rutina INCOME. Examina los chips USART
               9500 ;   * para ver si han recibido algún carácter, y actualiza
               9600 ;   * los contadores de error y los datos de la memoria
               9700 ;   * intermedia en uno de los 4 memorias intermedias
               9800 ;   * dedicadas - ver texto para los detalles.
               9900
              10000 ;   ** Versión 2b **
6562 F5       10100 INCOME PUSH AF    ; Salvar los registros que se van a usar
6563 C5       10200        PUSH BC    ;                ^
6564 E5       10300        PUSH HL    ;                ^
6565 D5       10400        PUSH DE    ;                ^
6566 01E1F9   10500        LD   BC,#F9E1  ; BC 1er reg estado USART
6569 210060   10600        LD   HL,#6000  ; HL base memoria intermedia
656C 1604     10700        LD   D,#4      ; contador bucle
656E ED78     10800 INC001 IN   A,(C)     ; Obtiene estado del USART
6570 0B       10900        DEC  BC        ; BC registro de datos
6571 FEFF     11000        CP   #FF       ; Si ha leido FF - no hay USART
6573 2824     11100        JR   Z,INC003  ; salta si no hay USART instalada
6575 CB4F     11200        BIT  1,A       ; Ver si está puesto RX-RDY
6577 2820     11300        JR   Z,INC003  ; si no, saltar
6579 E638     11400        AND  #38       ; Ver si hay bits de error
657B 2030     11500        JR   NZ,ERROR  ; Llama al registro de errores
657D 7D       11600 FNDEOB LD   A,L       ; Toma octeto inferior del apunt.
657E C603     11700        ADD  A,#3      ; en A. Súmale 3
6580 6F       11800        LD   L,A       ; Volverlo a poner modificado
6581 CB7E     11900 INC004 BIT  7,(HL)    ; ver si está puesto el bit alto
6583 2003     12000        JR   NZ,INC002 ; si no, saltar - no EOB así que
6585 23       12100        INC  HL        ; incrementa HL y prueba de nuevo
6586 18F9     12200        JR   INC004    ; bucle hasta encontrar EOB (Hex FF)
6588 ED78     12300 INC002 IN   A,(C)     ; Obtener el octeto
658A E67F     12310        AND  #7F       ; Limitar a ASCII
658C 77       12400        LD   (HL),A    ; Almacena el octeto
658D 23       12500        INC  HL        ; HL apunta el nuevo EOB
658E 36FF     12600        LD   (HL),#FF  ; Escribe el nuevo EOB
6590 3EFF     12700        LD   A,#FF     ; A=#FF
6592 BD       12800        CP   L         ; Ver si HL tiene rebasam. mem. int.
6593 2004     12900        JR   NZ,INC003 ; saltar si no
6595 2E03     13000        LD   L,#03     ; apuntador mem. int. a base
6597 36FF     13100        LD   (HL),#FF  ; Escribir nuevo EOB - borrar mem int
              13200 ;  Esto fuerza a la memoria intermedia a ser circular
              13300 ;  si hay rebosamiento.
```

```
6599  15      13400 INC003 DEC  D          ; Decrementar D
659A  2005    13500       JR   NZ,INC005  ; Saltar si queda algun USART
659C  D1      13600       POP  DE
659D  E1      13700       POP  HL
659E  C1      13800       POP  BC
659F  F1      13900       POP  AF         ; Limpiar la pila y .....
65A0  C9      14000       RET         ; volver sin cambios en los registros
              14100
65A1  2E00    14200 INC005 LD   L,#0       ; Cero octeto inferior de HL
65A3  7C      14300       LD   A,H        ; Mover H a A
65A4  C601    14400       ADD  A,#1       ; Sumar #100 a HL
65A6  67      14500       LD   H,A        ; reponer el octeto superior
65A7  79      14600       LD   A,C
65A8  C603    14700       ADD  A,#3       ; BC apunta sigui. reg. estado
65AA  4F      14800       LD   C,A        ; que apunta al siguiente USART
65AB  18C1    14900       JR   INC001     ; y repetir de nuevo.
65AD  CB5F    15000 ERROR BIT  3,A        ; Ver si hay error de paridad
65AF  2801    15100       JR   Z,OVERCK   ; si no, salta a sig. comprobación
65B1  34      15200       INC  (HL)       ; Incrementar contador err. paridad
65B2  23      15300 OVERCK INC  HL         ; HL apunta al siguiente contador
65B3  CB67    15400       BIT  4,A        ; ver si hay error de sobrecarga
65B5  2801    15500       JR   Z,FECHK    ; salta si no - o incrementa
65B7  34      15600       INC  (HL)       ; el contador de error sobrecarga
65B8  23      15700 FECHK  INC  HL         ; HL apunta al siguiente contador
65B9  CB6F    15800       BIT  5,A        ; ver si hay error de estructura
65BB  34      15900       INC  (HL)       ; incrementa el contador siguiente
65BC  2801    16000       JR   Z,ERR1     ; si no, terminado
65BE  34      16100       INC  (HL)       ; incrementa el contador FE
65BF  3E15    16200 ERR1   LD   A,#15      ; Carga código restaurar errores
65C1  03      16210       INC  BC         ; BC apunta a señalizadores del
65C2  ED79    16300       OUT  (C),A      ; registro de estado, y lo manda
65C4  0B      16310       DEC  BC         ; BC a registro de datos
65C5  18D2    16400       JR   INC003     ; y volvemos
              16500 ;   * Ahora se limpia el estado, el USART puede recibir ya
              16600 ;   * otro carácter
              16700
Pass 2 errors: 00

ERR1   65BF  ERROR  65AD  FECHK  65B8  FNDEOB 657D
GOUT   655F  INC001 656E  INC002 6588  INC003 6599
INC004 6581  INC005 65A1  INCOME 6562  K1     6507
KERNEL 6500  KEYCHK 651F  OVERCK 6582  SEND   6532
SEND01 653D  SHUF01 6554  SHUFFL 6548

Table used:   248  from   878
Executes: 25856
```

Fig 3.55 Listado del ensamblador de MACCODE

Los módulo de MACCODE en detalle

Si no sabe nada de programación en lenguaje ensamblador, sáltese esta sección y vuelva a ella cuando haya leído el capítulo cinco. La figura 3.55 nos muestra el listado del ensamblaje de la sección MACCODE del controlador serie. Los siguientes párrafos nos dan una breve descripción de las acciones y decisiones tomadas por cada módulo:

KERNEL: Como sugiere su nombre KERNEL es la propia rutina de la sección de código máquina. Cuando se ejecuta hace una llamada al sistema para hacer que aparezca el cursor en la pantalla. (El BASIC siempre elimina el cursor cuando se está ejecutando). A continuación, KERNEL llama a otra de las rutinas que la acompañan, INCOME. Esta rutina se describirá enseguida, pero examina todos los USART para ver si ha llegado algún carácter a sus memorias intermedias. Después KERNEL ejecuta una pequeña secuencia para comprobar si hay caracteres pendientes en el canal seleccionado, si los hay los imprime en la pantalla del CPC usando otra llamada a la ROM. Después de mostrar el carácter en la pantalla, KERNEL procede a llamar a otra de las rutinas que la acompañan - SHUFFL. Esta rutina mueve el contenido de la memoria intermedia del canal seleccionado, un octeto hacia arriba. Finalmente, KERNEL comprueba si se ha tecleado algún carácter en el teclado del CPC. Si no se ha tecleado nada, se reanuda a sí misma, si se ha tecleado, comprueba si el código del carácter es el correspondiente a CTRL/P o CLR. Si lo es, vuelve al BASIC. Si no, llama a la rutina SEND, y cuando vuelve se restaura la rutina KERNEL.

SEND: Esta rutina manda al canal seleccionadoel carácter que le llega en el registro A. Toma el número del canal seleccionado, y lo convierte en la dirección de un USART. Después espera a que esté preparada la sección de trasmisión del USART y manda el carácter.

SHUFFLE: Es una rutina que desplaza todos los caracteres de la memoria intermedia de recepción un carácter hacia adelante. La usan todas las rutinas que toman un carácter del principio de la memoria intermedia, para mover los demás caracteres hacia adelante, y deja sitio al final de la memoria intermedia. De esta forma, cada vez que se toma un carácter de la memoria intermedia para sacarlo a la pantalla, el resto de los caracteres se mueven hacia arriba, y se queda un octeto disponible al final de la misma.

INCOME: La rutina INCOME es llamada desde la rutina KERNEL para comprobar si han llegado nuevos octetos de datos al USART. Es la rutina que actualiza los contadores de error de cada canal. Su propósito es proporcionar los contadores de error para permitirle comprobar el número de errores de datos que ocurren durante la recepción. Una cantidad excesiva de errores de datos indicará un problema, de 'hardware' o de línea, que debe ser solucionado. Estos errores ocurren siempre en aplicaciones reales, pero uno o dos debe ser el máximo tolerado.

INCOME toma los caracteres recibidos correctamente y los coloca en la memoria intermedia apropiada. El nuevo carácter reemplaza el marcador EOB - valor Hex FF - e INCOME escribe un nuevo EOB en el octeto detrás del nuevo carácter, a menos que la memoria intermedia se haya llenado, en cuyo caso se reinicializa y se pierde el contenido de la misma. Los lectores que necesiten memorias intermedias de recepción mayores no deben encontrar ninguna dificultad para modificar el programa y ajustarlo a sus necesidades.

Ya que INCOME comprueba continuamente todos los USART instalados, no se deben perder caracteres. El 'hardware' y el 'software' han sido comprobados conectando los cuatro canales a cuatro líneas serie de un super mini ordenador VAX y mandando simultáneamente un fichero de 200 octetos a los cuatro canales, a una velocidad de 2400 baudios. Esta conexión funcionó sin problemas, por lo que tampoco debe haberlos en aplicaciones menos críticas.

Puntos generales

Ya que el interfase de cuatro canales es bastante complejo para su CPC, hay un par de puntos que debe tener en cuenta cuando lo use.

El primero es que el hecho de que haya dos canales conectados a cada generador de velocidad de trasmisión significa que, cuando cambie ésta en un canal, también cambiará la del otro. Cuando esté usando el programa controlador serie, la velocidad de trasmisión de un par de canales será la seleccionada más recientemente en el menú de opciones. Por ejemplo, si va al menú de opciones y elige el canal 3 como canal seleccionado y después lo pone a 300 baudios, el controlador pone también el canal dos a 300 baudios. Esto puede dar como resultado errores, si el canal dos está conectado a una línea de diferente velocidad. Por lo tanto tenga en cuenta este tipo de cosas si empieza a sufrir muchos errores. Usted debe conectar cada par de canales a canales con la misma velocidad de trasmisión, pero puede que haya ciertas situaciones en que esto no sea posible.

Si usted usa el interfase para poder usar su CPC como una unidad VDU multicanal, no debería tener problemas de mensajes perdidos. Si lo usa como una VDU con tres canales de mensajes, deberá entrar regularmente en el menú de opciones para asegurarse de que no hay mensajes esperando en los otros canales, si no lo hace, perderá algunos mensajes cuando se llenen las memorias intermedias.

Cuando esté recibiendo grandes cantidades de datos a altas velocidades pueden sucederle errores de sobrecarga en el canal seleccionado. Esto puede ocurrir cuando tenga que mover el texto de la pantalla por medio de rutinas de la ROM del CPC. Parece ser que las rutinas de la ROM retienen el control del procesador durante el tiempo en que maneja la pantalla, y por lo tanto INCOME no tiene oportunidad de recoger los caracteres recibidos, dando como resultado errores de sobrecarga. Solamente he experimentado este defecto cuando se mandan saltos de línea comtinuos a velocidades mayores de 2400 baudios, y no afectaron a la operación normal. Si ocurre este tipo de error, el señalizador se restaura y se pierde un carácter.

Algunas aplicaciones típicas

Hay varias configuraciones típicas en las que se puede usar la tarjeta de cuatro canales RS232. Puede ser útil el enumerarlas.

1) Usar el CPC como un VDU multicanal en un establecimiento comercial o educativo conectado a algunos ordenadores grandes. La figura 3.56 muestra cómo se debe conectar para este tipo de aplicación.

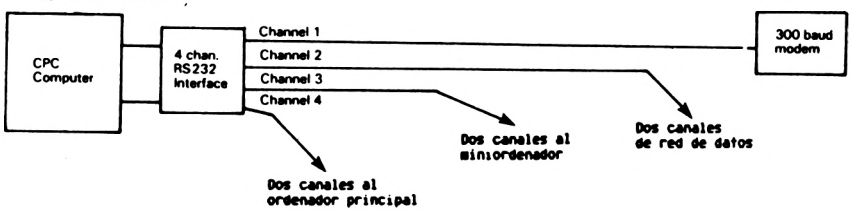


Fig 3.56 Diagrama de bloques que muestra el uso del CPC y el interface de cuatro canales RS232 como unidad de visualización (VDU) multicanal.

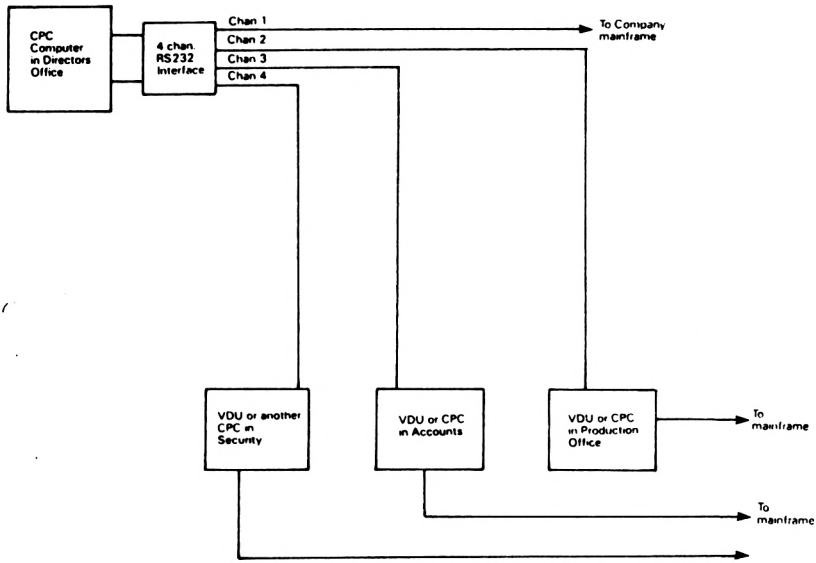


Fig 3.57 Uso del CPC y el interfase de cuatro canales como una estación de recepción de mensajes y VDU

2) Usar el CPC como una VDU, pero con capacidad de recepción de mensajes desde otro VDU o CPC con un equipo similar. Esto nos permitirá usar el CPC como una VDU normal, pero retener los mensajes mandados desde otras oficinas del mismo edificio, para poderlos ver después. La figura 3.57 muestra cómo se puede realizar esta conexión.

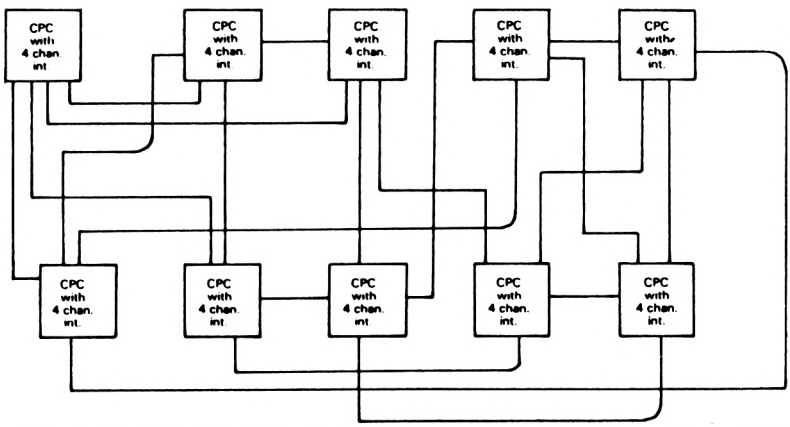


Fig 3.58 Uso del CPC y el interfase de cuatro canales como un terminal de mensajes

3) Usar el CPC como un terminal de recepción de mensajes, como rercordatorio de intercomunicaciones. Esto es, si lo tiene en la pantalla es difícil que se le olvide. Ver figura 3.58.

Hoja informativa de la Versión B del RS232

Tiempo empleado por todas las velocidades posibles para trasmitir 2048 octetos

---

19200	Baudios tarda	1.2	segundos
50	Baudios tarda 7 minutos y	5	segundos
75	Baudios tarda 5 minutos y	0.5	segundos
134.5	Baudios tarda 2 minutos y	9	segundos
200	Baudios tarda 1 minuto y	9	segundos
600	Baudios tarda	37.6	segundos
2400	Baudios tarda	9.4	segundos
9600	Baudios tarda	2.3	segundos
4800	Baudios tarda	4.7	segundos
1800	Baudios tarda	12.6	segundos
1200	Baudios tarda	19	segundos
300	Baudios tarda 1 minuto y	15.2	segundos
150	Baudios tarda 2 minutos y	30.5	segundos
110	Baudios tarda 3 minutos y	27.5	segundos

--

Tabla que muestra las velocidades disponibles, usando el divisor por 1, 16 o 64 del USART 8251A. Esta información puede ayudarle a obtener dos velocidades requeridas en USART conectados al mismo generador de velocidades de trasmisión. Por ejemplo, si va a cargar seis en el registro de velocidad de trasmisión del USART 0, puede programar el USART 1 para dividir por 16 y hacer que el canal 1 funcione a 600 baudios, y programar el USART 2 para dividir por 64 para tener un canal 2 funcionando a 150 baudios.

Valor en Baud. registro veloc.	x 1	x 16	x 64
0	307200	19200 *	4800 *
1	307200	19200 *	4800 *
2	800	50 *	12.5
3	1200 *	75 *	18.75
4	2152	134.5 *	33.625
5	3200	200 *	50 *
6	9600 *	600 *	150 *
7	38400 *	2400 *	600 *
8	153600	9600 *	2400 *
9	76800	4800 *	1200 *
A	28800	1800 *	43.75 *
B	19200	1200 *	300 *
C	38400	2400 *	600 *
D	4800 *	300 *	75 *
E	2400 *	150 *	37.5 *
F	1760	110 *	27.5 *

* señala una velocidad estándar

Fig 3.59 Hoja informativa del inteface RS232 VB de cuatro canales

4) Como equipo de pruebas multicanal. Se puede usar como terminal de una tienda de reparación, donde se arreglan impresoras o VDU. El CPC se puede usar como consola para mandar mensajes a las unidades bajo prueba. Con un poco más de 'software' (Modificaciones a la rutina KERNEL) puede hacer que el CPC mande grupos continuos de caracteres a todos los canales. (Puede modificar también el programa de pruebas para hacerlo).

5) Para manejar una impresora con protocolo serie, que necesite 8 bits de datos y por lo tanto no pueda funcionar con la puerta centronics del CPC.

Es posible que alguien en algún lugar encuentre más aplicaciones, pero estas serán quizás las más útiles.

Conclusión

La versión B del interface RS232 representa una pieza de 'hardware' sumamente útil para su CPC. Le da la oportunidad de interconectarse con una gran variedad de dispositivos, y le abre paso a un mundo de redes de comunicación, comunicaciones entre ordenadores, y toda clase de trasferencia de datos. La figura 3.59 es una tabla informativa sobre la versión B del interfase, que le muestra las velocidades de trasmisión que puede obtener, usando la opción de división por 64 del 8251A, por medio de las opciones por defecto de programa controlador. También le muestra una tabla de tiempos para trasferir 2K octetos de datos a las velocidades más usadas. La figura 3.60 le muestra las patillas de alimentación de los chips del proyecto y la figura 3.61 es la lista final de componentes. Finalmente, los lectores deben referirse al apéndice cuatro que les muestra las conexiones RS232, y al apéndice seis para una descripción de la trasmisión serie.

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	OV
1	74LS138	Decodificador 3 a 8 líneas	16	--	--	8
2	74LS02	4 puertas NOR 2 entradas	14	--	--	7
3	74LS21	2 puertas AND 4 entradas	14	--	--	7
4	4702B	Gener velocidad trasmisión	16	--	--	8
5	4702B	Gener velocidad trasmisión	16	--	--	8
6	8251A	Chip USART	26	--	--	4
7	8251A	Chip USART	26	--	--	4
8	8251A	Chip USART	26	--	--	4
9	8251A	Chip USART	26	--	--	4

Fig 3.60 Patillas de alimentación de los chips del RS232 VB

PROYECTO 8: PROGRAMADOR DE EPROM

Una de las cosas más útiles que se pueden tener, cuando se expande el 'hardware' de un ordenador personal, es sin duda, un programador de EPROM. Si usted sabe realmente lo que es un programador de EPROM y cómo usarlo, sáltese estos dos párrafos siguientes.

EPROM es la abreviatura de Erasable Programable Read Only Memory (Memoria de solo lectura borrable y programable). Como ya sabrá, hay dos tipos de chips de memoria para ordenadores, RAM o Random Access Memory (Memoria de acceso aleatorio) que pierde su contenido cuando se le corta la alimentación, y ROM o Read Only Memory (Memoria de solo lectura) que mantiene su contenido para siempre. La EPROM está más cerca de la ROM que de la RAM, pero a diferencia de la ROM se pueden escribir datos en ella. La forma en que funciona la clase de EPROM con la que vamos a trabajar es aplicándole un voltaje alto especial, para ponerlo en un modo en el que, con el soporte de algún circuito lógico, podemos introducir datos dentro de ella.

Lista de componentes de la versión B del interfase RS232 de cuatro canales

Resistencias

R1	10M Ohmios 1/2 watio
R2-R5	5K6 Ohmios 1/2 watio

Condensadores

C1-C2	56 pF 16 voltios
C3-C6	470 pF 16 voltios

Semiconductores

E1	74LS138 decodificador de 3 a 8 líneas	
E2	74LS02 cuatro puertas NOR de dos entradas	
E3	74LS21 dos puertas AND de cuatro entradas	
E4-E5	4702B generador de velocidad de trasmisión	**
E6-E9	8251A chip USART	**

Deben ser 8251A: La mayoría de los 8251 no funcionan correctamente

E10-E11	74LS174 chip de retención hexadecimal	**
E12	1488 chip transmisor RS232	
E13	1489 chip receptor RS232	

Varios

Zócalos para todos los chips
Circuito impreso
XTAL 1 = Cristal oscilador de 2.45676 MHertz
Conector DIL de 16 patillas

NOTAS

1) De los componentes marcados con dos asteriscos no se requieren todas las unidades si no va a construir todos los canales. La siguiente tabla si/no le indica los componentes necesarios dependiendo del número de canales que vaya a instalar.

	Número de circuito integrado							
	E4	E5	E6	E7	E8	E9	E10	E11
Canales 2	Si	No	Si	Si	No	No	Si	No
Instalados 3	Si	Si	Si	Si	Si	No	Si	Si
4	Si	Si	Si	Si	Si	Si	Si	Si

Fig 3.61 Lista de componentes para el proyecto RS232 VB

La secuencia exacta es esta: primero se aplica la tensión de programación a la patilla VPP. En el caso de una EPROM grande (como las dos para las que se describe el programador) esta tensión es de 21 voltios de corriente continua, para EPROM más pequeñas suele ser de 25 voltios. Después se aplica la dirección a las patillas de dirección de la EPROM. A continuación se colocan en las líneas de datos los datos que queremos introducir. Después se aplica un nivel bajo TTL durante 50 milisegundos (0.050 segundos) a la patilla PGM del chip. Cuando hemos terminado, se liberan las líneas de dirección y de datos, y el octeto ha sido programado con el dato. Estos pasos se repiten hasta que se han programado el número de octetos deseado. Se pueden programar cualquier número de octetos de una EPROM, no es necesario programar todo su contenido. Una vez que se han programado los octetos deseados, el chip pasa a ser como una ROM, manteniendo su contenido intacto aunque se quede sin alimentación. En la otra cosa en que difiere una EPROM de una ROM es en que puede ser borrada usando ciertas fuentes de luz Ultra Violeta. Todos los chips EPROM tienen una pequeña ventana en lo alto que permite que entre dentro del chip la luz UV de ciertas lámparas especiales. La EPROM solo se puede programar y borrar un cierto número de veces, antes de que se inutilice.

Los chips EPROM se usan en una gran cantidad de aplicaciones, donde su bajo precio y el hecho de que sean reusables, hacen que no tengan competencia con los chips ROM. Su existencia hace posible que los no profesionales y los fabricantes en pequeñas cantidades produzcan unidades que no podrían ser tan baratas si no existieran este tipo de chips. Los ordenadores Amstrad le permiten añadir una expansión en ROM que proporciona programas extras accesibles al instante. Una expansión ROM puede ser, como veremos, una EPROM con sus propios programas extra dentro de ella.

Actualmente hay gran cantidad de chips EPROM disponibles en el mercado, pero hay dos en particular que suelen tener en casi todos los establecimientos de electrónica, y son el 2764 y el 27128. Estos chips son para los que está diseñado este proyecto. Ambos dispositivos tienen 28 patillas y está contenidos en una cápsula DIL, y tienen todas las conexiones idénticas, excepto la patilla 26 que no está conectada en el 2764, y se usa como línea de dirección 13 en el 27128. El 2764 tiene 8192 posiciones, cada una de ocho bits. El 27128 tiene 16384 posiciones, también de ocho bits cada una. La figura 3.62 es una reproducción de la hoja de especificaciones de Intel Corporation, que es la fuente principal de fabricación del 27128, el 2764, y de todos los otros chips de la familia 27XXX. La figura 3.62 es especialmente útil debido a que nos muestra las distribución de las patillas de todos los chips EPROM de la familia 27XXX desde el 2716 en adelante.

Hay muchas formas diferentes de construir un programador de EPROM. Sin embargo, la más simple es introducir los datos en la EPROM por medio de interruptores, usando el código binario requerido para cada octeto. Este método fue aceptable en los días del chip 2708 (1Kx8 bits), pero como ahora estamos tratando con chips con capacidad 16 veces superior, la cantidad de datos que se deben introducir dentro de la EPROM hace poco práctico este sistema.

El segundo método se basa en el hecho de que en ciertos ordenadores parte del espacio direccionable no está ocupado, ubicando en esta memoria el contenido de la EPROM. Esto era adecuado cuando el tamaño de las EPROM y de la memoria de los ordenadores personales era pequeña,

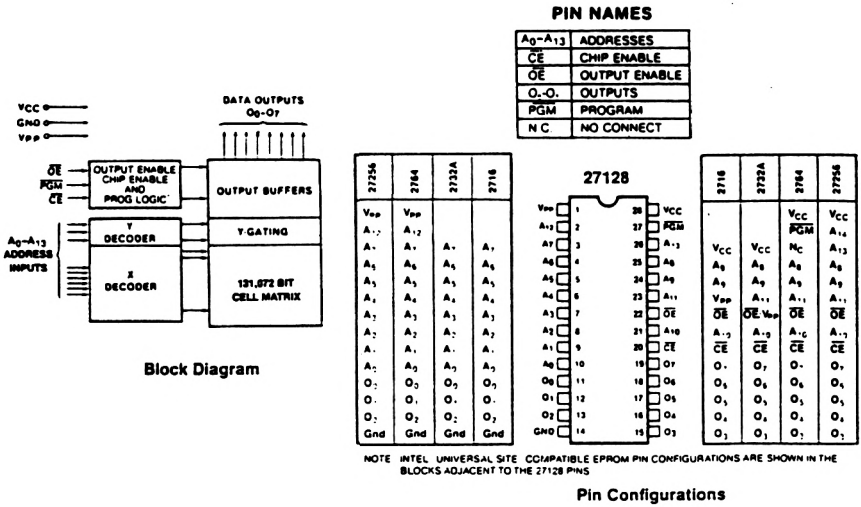


Fig 3.62 Hoja de especificaciones de las EPROM 27xx
(Traducción en la siguiente página)

Reproducida con el permiso de Intel Corp.

27128

PROM BORRABLE POR UV 128K (16K x 8)

Tiempo Máximo de Acceso, 250 ns...
Tecnología HMOS*-E

Compatible con la MPU iAPX de Alta velocidad a 8 MHz..Estado WAIT cero

Control por Dos-Líneas

Patillas compatibles con la EPROM 2764

Patillas Estándar Industrial ...Aprobado por JEDEC

Tolerancia de Tensión VCC 10%

Características de Baja Potencia:
-Corriente Activa Max 100 mA
-Corriente Alerta Max 40 mA

El Intel 27128 es una memoria de solo lectura, programable electricamente y borrable por ultravioletas (EPROM), de 131,072 bits alimentada solamente por 5V. El tiempo de acceso estándar del 27128 es de 250 ns, lo que la hace compatible con los microprocesadores de alto rendimiento, como el Intel iAPX de 8 MHz. En estos sistemas, el 27128 permite al microprocesador opera sin la adición de estados de WAIT.

Una característica importante de la 27128 es el control de salida separado, Output Enable (OE) desde el control de Chip Enable (CE). El control OE elimina la contención del 'bus' en los sistemas microprocesadores de 'bus' múltiple. La Nota de Aplicación AP-72 de Intel, describe la implementación en el sistema microprocesador de los controles OE y CE de las EPROM de Intel. AP-72 está disponible en los departamentos de literatura de Intel.

El 27128 tiene un modo de alerta que reduce la disipación de potencia sin incrementar el tiempo de acceso. La corriente activa es de 100 mA, mientras que la corriente de alerta es de solo 40 mA. El modo alerta se consigue mediante la aplicación de una señal TTL a la entrada CE.

La tolerancia de Vcc del 10% en el 27128, es una alternativa a la estándar de 5V. Esto permite al diseñador del sistema más libertad en los requerimientos de la fuente de alimentación y en otros parámetros del sistema.

El 27128 se fabrica con tecnología HMOS*-E. Tecnología Intel de alta velocidad de puertas MOS de silicona de N canales.

*HMOS es una patente de Intel Corporation.

Traducción de la hoja de características de la Fig 3.62

pero ahora podemos tener ciertas dificultades en encontrar 16K de memoria sin usar. En las máquinas CPC se puede intercambiar el espacio de la ROM superior - sustituyendo la EPROM por el BASIC, pero esto haría que el programa se complicara, ya que usted no puede leer normalmente la 16K superiores de la ROM para comprobar que no se han introducido errores al programar la EPROM, pero no obstante se puede hacer.

Otro método de diseño de programador de EPROM, usa una conexión entre su ordenador y una unidad de programación para trasferir los datos a la EPROM. (Este método puede ser usado en el CPC, haciendo uso del canal de trasferencia paralela descrito anteriormente). Un microprocesador, o similar, hace la programación dentro de la unidad externa. Este método es bueno cuando usted necesita ahorrar tiempo del procesador, como en un sistema multi-usuario, o multi-tarea, pero requiere bastante 'hardware' exterior que no es justificable cuando el procesador no tiene otra cosa que hacer mientras la memoria EPROM se está programando.

El método final es el que se ha elegido para nuestro proyecto de programador de EPROM. Este método incluye el uso de otro chip PPI, cuyas puertas están conectadas a las líneas de datos y de direcciones de la EPROM que se va a programar. Como es usual, el PPI aparece ante el sistema como cuatro registros de E/S. Un quinto registro, de solo lectura, suministra información del estado del programador de EPROM. Las direcciones de los registros y las imágenes de los registros de lectura/escritura se muestran, junto con un diagrama de bloques del programador, en la figura 3.63. El concepto de BUS LOCAL (o 'bus' paralelo) a que se refiere el diagrama se usa también en el siguiente proyecto.

Las ventajas de esta aproximación a un diseño de programador de EPROM tienen dos vertientes. Primero, siempre existe la pequeña posibilidad de que ocurran accidentes con la EPROM grabada directamente. Puede ser que deje la tensión de programación conectada, e introduzca basura inadvertidamente dentro de la EPROM, o que el 'hardware' tenga algún error
e introduzca una alta tensión en el 'bus' del Z80. En este esquema el PPI le protege en gran manera al resto del ordenador. No suele suceder, pero es posible. La mayor ventaja de este método es que tiene que asignar solo cinco direcciones de E/S al programador. Esto elimina la necesidad de eliminar algo de la memoria para hacerle sitio a la EPROM. No es demasiado rápido tener que cargar primero el octeto bajo de la dirección, después el octeto alto y luego el dato dentro del PPI, pero programar una EPROM de 8K o 16K es una tarea realmente lenta. (Ocupa en total. 0.050 x 16384 = 819 segundos - o trece minutos - y solo en los impulsos de programación).

En la figura 3.64 se muestra el esquema del decodificador de direcciones y de la lógica de control del circuito del programador de EPROM. Los registros tiene direcciones desde Hex F8E0 a F8E7. (En el diagrama se muestran las máscaras de bits de estas direcciones). Como con todos los periféricos de expansión del 'bus', los cuatro bits superiores de dirección se pueden poner con cualquier valor, siempre que el bit A10 esté a cero, y de esta forma A12-A15 no se decodifican. La dirección es decodificada por E1a y E2a, que dan paso al decodificador E3. En este diseño solo se usan dos de las salidas del decodificador, pero se puede adaptar si se quiere construir un segundo PPI y zócalo de programación para convertir este proyecto en un

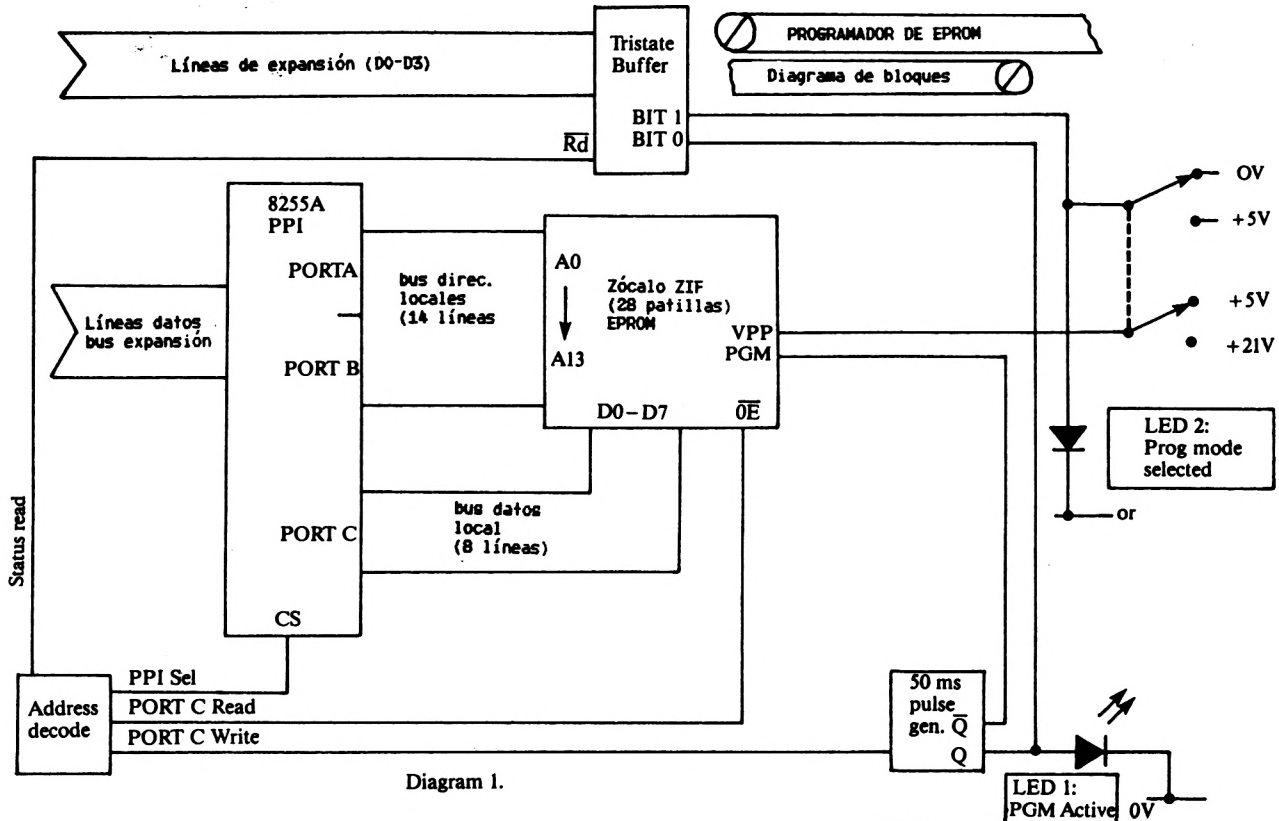


Fig 3.63 Diagrama de bloques del programador de EPROM

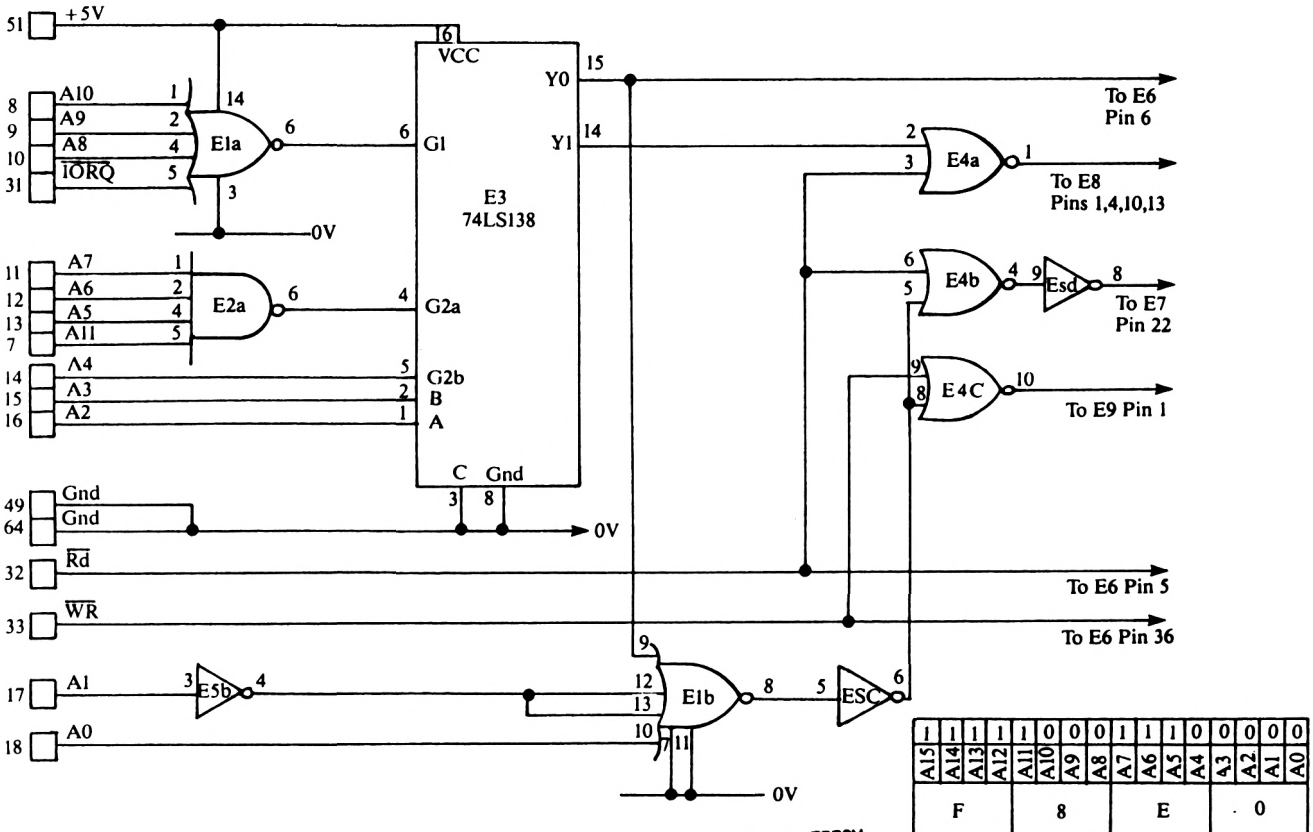


Fig 3.64 Decodificador de direcciones y circuito de control del programador de EPROM

copiador de EPROM. Las líneas de dirección A2 y A3 se introducen en E3 y se usan para decodificar cuándo se puede direccionar el registro de estado del PPI. Si el PPI se está direccionando, la lógica interna de decodificación del 8255A selecciona el registro apropiado dentro del PPI. Las tres puertas del PPI se usan como sigue:

PUERTA A - Siempre como puerta de salida - conectada a las 8 líneas inferiores de dirección de la EPROM que se está programando.

PUERTA B - Siempre como puerta de salida - conectada a las 8 líneas superiores de dirección de la EPROM que se está programando.

PUERTA C - Se selecciona como puerta de salida cuando se programa una EPROM, y como entrada cuando se comprueba su contenido. La puerta C se conecta a las líneas de datos de la EPROM que se está programando.

La puerta E1b detecta cuándo se está accediendo a la puerta C y las puertas E4b y E4c dispararán el generador de impulso de programación si el acceso es para escribir, o activarán la salida de la EPROM si el acceso es para leer. La puerta E4a activa el registro de estado si ocurre una lectura desde una dirección de E/S en el rango F8E4 a F8E7.

La figura 3.65 muestra el chip PPI - E6 - y el zócalo de la EPROM. Este zócalo es un 'Zero Insertion Forze - ZIF - (Fuerza de Inserción Cero). Estos zócalos tienen una palanca especial que libera las patillas del chip dentro del zócalo, evitando el problema de que se puedan doblar las patillas al meter y sacar el chip. La figura 3.65 muestra también el conmutador de selección de LECTURA/PROGRAMACION. Es un conmutador de dos circuitos dos posiciones. Los dos circuitos separados son de suma importancia. Uno se usa para introducir un nivel lógico cero o uno (OV o +5V) dentro del registro de estado, y también tiene conectada la luz de aviso de modo programación (un LED intermitente). El segundo circuito se usa para conectar los +21 voltios o los +5 voltios a la entrada VPP de la EPROM. Los impulsos de programación que vienen del generador de impulsos de programación (ver figura 3.66) se introducen en el zócalo ZIF de la EPROM por la patilla 27 - que es la entrada PGM. Obviamente se debe tener mucho cuidado de no puentear los dos circuitos del conmutador. Finalmente, el inversor E5e invierte el impulso de restauración del 'bus' a la polaridad requerida por el 8255A. El chip PPI es seleccionado mediante la señal aplicada a su patilla 6 desde el circuito decodificador de direcciones. Como se muestra en el esquema, el octeto de modo del PPI debe ser Hex 80, cuando se está programando la EPROM, y Hex 89 para los demás usos.

La figura 3.66 muestra los restantes elementos de 'hardware'. Estos son el registro de lectura de estado y el generador de impulsos de programación. El registro de lectura de estado lo usan los programas para asegurarse de dos cosas: Si el generador de impulsos de programación está activo, y qué posición tiene el conmutador de LECTURA/PROGRAMACION. La imagen de los registros aparece dibujada en la figura 3.66. El generador de impulsos de programación está constituido por un monoestable, con la longitud del impulso de salida, ajustable mediante VR1, entre 45 y 60 milisegundos. (Más adelante se dan detalles de ajuste). Sáltese el siguiente párrafo si realmente conoce lo que es un monoestable y cómo se usa.

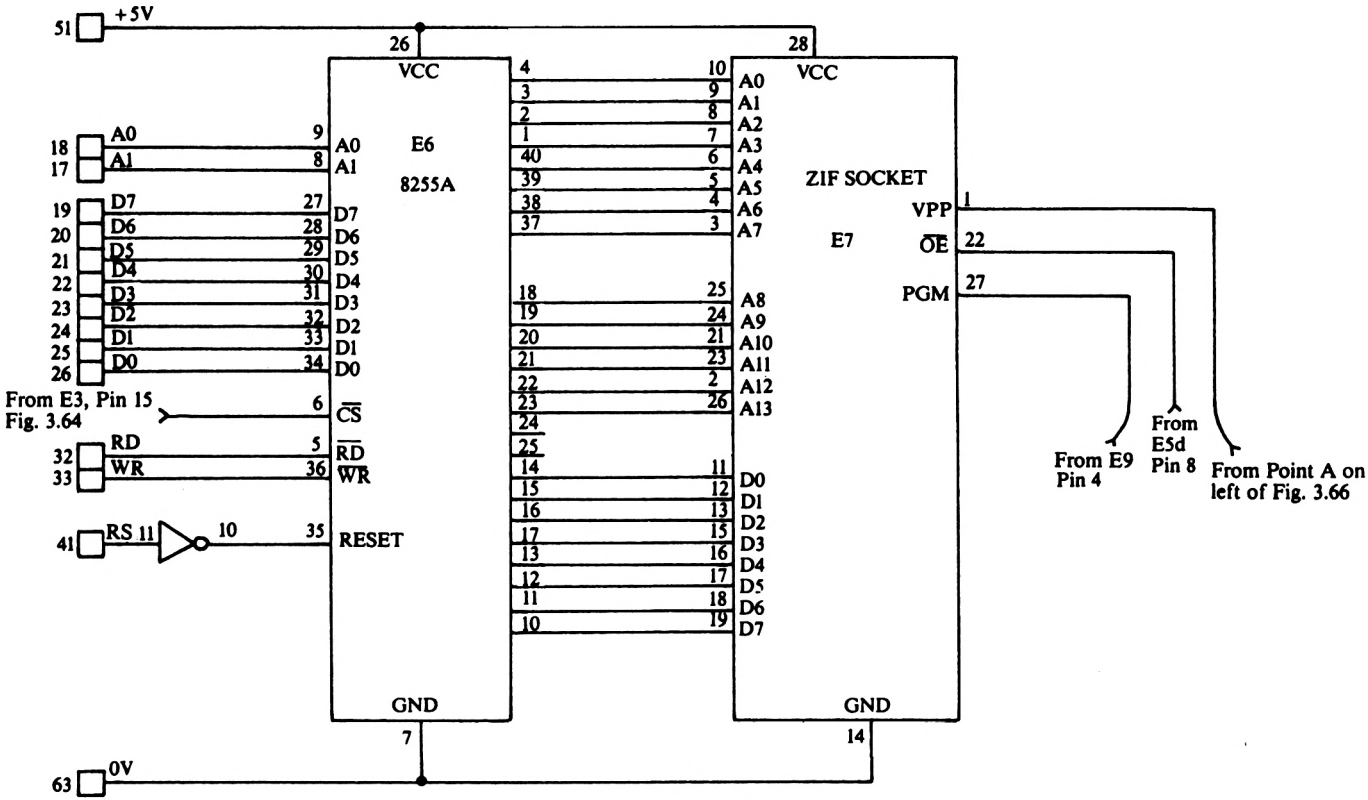


Fig 3.65 Chip PPI y zócalo ZIF del programador de EPROM

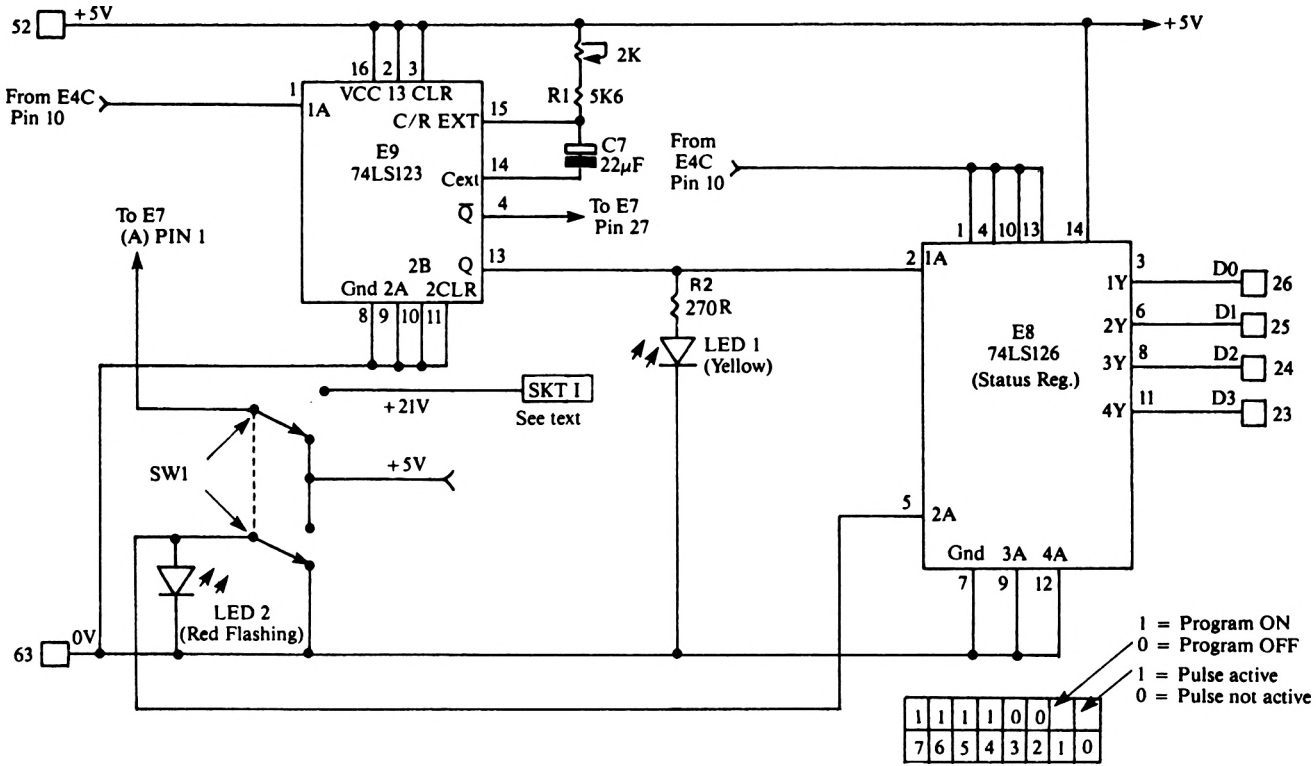


Fig 3.66 Resto del circuito del programador de EPROM

Gran parte de la electrónica de los ordenadores depende de la generación y detección de impulsos de tensión. Un monoestable es un circuito, disponible en un chip como el 74LS123, que se usa para generar impulsos.

Un monoestable suele tener dos salidas. Estas son las salidas Q y Q negativa. La salida Q está normalmente a nivel bajo, pero se pone a nivel alto cuando se dispara. La salida Q negativa está normalmente a nivel alto pero se pone a nivel bajo cuando se dispara el monoestable. La duración del impulso, que sale cuando ocurre la condición de disparo, se puede ajustar mediante la conexión de un condensador y una resistencia a las patillas de temporización CEXT y RC/EXT del monoestable. Como en este caso, la longitud del impulso se puede hacer ajustable haciendo que la resistencia sea variable. Las condiciones requeridas para disparar el chip 74LS123 (que contiene dos monoestables separados - uno no se usa en este proyecto) puede ser un impulso que pasa a negativo, o a positivo. Para una descripción más completa del 74LS123 consulte un libro de datos TTL.

El LED amarillo "Impulso de programación activo" se conecta por medio de una resistencia a la patilla de salida Q del monoestable. Esta patilla va a parar también a la entrada de E8 que es un chip de 'buffer' tri-estado usado como registro de estado. El monoestable es disparado por un borde negativo en su entrada A. Esto sucede cuando se escribe en la puerta C, y la puerta E4c (mostrado en la figura 3.63) manda su impulso sobre la patilla 1 del monoestable. Esto dispara el monoestable, y la salida Q negativa manda un impulso a la entrada PGM de la EPROM, como ya se ha descrito.

Antes de ver los programas para hacer toda la preparación y manejo del programador, he aquí una explicación detallada paso a paso de como se programa un octeto dentro de la EPROM en nuestro diseño.

1) Coloque la EPROM dentro del zócalo ZIF y cierre la palanca de retención. Asegúrese de que el conmutador de LECTURA/ESCRITURA está en posición de LECTURA. Conecte el cable de los 21 voltios desde la fuente de alimentación a PL1 en la tarjeta del programador.

2) Encienda la fuente de alimentación.

3) Inicialice el PPI con el octeto de modo Hex 80, usando el comando:

```
OUT &F8E3,&80
```

4) Ahora mande el octeto inferior de la dirección LOCAL a la puerta A:

```
OUT &F8E0,&00
```

5) A continuación mande el octeto superior de la dirección LOCAL a la puerta B:

```
OUT &F8E1,&00
```

6) Ahora coloque el conmutador de LECTURA/PROGRAMACION en posición PROGRAMACION. El LED indicador de PROGRAMACION empezará a parpadear.

7) Mande a la puerta C el valor a escribir dentro de la EPROM. Por ejemplo para programar el valor Hex C9 use:

```
OUT &F8E2,&C9
```

8) El LED de PROGRAMADOR ACTIVO (el amarillo) parpadeará brevemente. Ahora cambie el conmutador LECTURA/PROGRAMACION de nuevo a LECTURA.

9) Cambie el modo del PPI a Hex 89 tecleando:

```
OUT &F8E3,&89
```

10) Ahora vuelva a mandar la dirección LOCAL como se hizo en los pasos 4 y 5.(Esto se debe hacer porque las puertas del PPI se restauran cuando se cambia el modo).

11) Ahora teclee:

```
PRINT HEX$(INP(&F8E2))
```

que imprimirá el valor del octeto que acaba de programar dentro de la EPROM. (En el ejemplo usado en el paso 7 debe ser C9).

12) **Apague la máquina y desconecte PL1.**

¡Todo este trabajo para programar solamente un octeto! Lo que acabamos de hacer es un trabajo muy largo, ya que no hemos presentado el programa que hace todo esto por usted, pero espero que esta descripción paso a paso le haga entender más fácilmente el programa.

```
100   '** Proyecto de programador de EPROM: programa de preparación del
       ** monoestable. Este programa re-dispara el impulso del monoestable de
110   '** programación para permitirle a usted ajustar su duración usando un
       ** osciloscopio
150   ON BREAK GOSUB 32700: W=25
200   MODE 2: INPUT "Hay un chip EPROM en el programador?";R$:
      IF UPPER$(LEFT$(R$,2)) <> "NO" THEN CLS:
      PRINT "Apague, quite el chip y ejecute de nuevo el programa": GOTO 32767
300   PRINT "Pulse ENTER para incrementar el tiempo entre disparos":
      PRINT "o la barra de espaciado para disminuirlo":
      '** Esta es la forma de hacerlo
400   OUT &F8E2,&FF
410   FOR I=1 TO W: R$=INKEY$: IF R$="" THEN 420 ELSE
      IF ASC(R$)=13 THEN W=W+1 ELSE IF ASC(R$)=32 THEN W=W-1
420   NEXT I: GOTO 400
32700 MODE 2: PRINT "Programa abortado por **BREAK**"
32767 LOCATE 10,10: END
```

Fig 3.67 Programa para disparar continuamente el generador de impulsos

Ajustando la longitud del impulso de programación

El ajuste del circuito se hace mediante VR1, hasta conseguir una longitud del impulso de 50 milisegundos a la salida del monoestable: En la figura 3.67 le proporcionamos el listado de un programa que dispara continuamente el monoestable . Si tiene a mano un

osciloscopio será una tarea sencilla (la precisión debe ser de 50 milisegundos mas/menos 2 milisegundos). Si no tiene disponible un osciloscopio, deberá usar el programa listado en la figura 3.68. Este programa muestra en la pantalla cuántos ticks de 300 milisegundos han ocurrido mientras ha estado activo el impulso. No hay una relacción directa entre los ticks y los 50 milisegundos debido a los retrasos producidos durante el proceso del programa. Debe ejecutar este programa duarante un minuto por lo menos, para permitir la estabilización, y ajustar a continuación VR1 hasta que aparezcan los números 15 o 16 en la parte izquierda de la pantalla. (mejor 15). una vez hecho esto, el ajuste se ha completado.

```
100   '** Proyecto de programador de EPROM: programa de ajuste del monoestable.
      ** Este programa le permite ajustar la duración del impulso de programación
      ** sin usar el osciloscopio.
150   ON BREAK GOSUB 32700
200   MODE 2: INPUT "Hay un chip EPROM en el programador?";R$:
      IF UPPER$(LEFT$(R$,2)) <> "NO" THEN CLS:
      PRINT "Apague, quite el chip y ejecute de nuevo el programa": GOTO 32767
300   CLS: WINDOW #1,10,80,1,25: WINDOW #0,1,5,1,25: LOCATE #1,10,10:
      PRINT #1,"Ejecútelo durante 1 minuto y después ajústelo como se describe
      en el libro,": LOCATE #1,10,12: PRINT #1,"hasta conseguir 15 o 16"
400   OUT &F8E2,&FF: T=TIME: WHILE (INP(&F8E4) AND 1) <> 0: WEND:
      PRINT TIME-T: GOTO 400
32700 MODE 2: PRINT "Programa termiando por **BREAK**"
32767 LOCATE 10,10: END
```

Fig 3.68 Programa de ajuste del programador de EPROM

Programas para manejar el programador de EPROM

El programa controlador del programador escribirá dentro de la EPROM el contenido de un área de RAM, o datos contenido en un fichero ASCII. (Por ejemplo un programa BASIC salvado usando el formato ",A", o un fichero creado usando AMSWORD u otro procesador de textos). El programa puede avisar si se le acaba la EPROM antes de terminar de escribir los datos, y lanzará un mensaje para permitirle colocar otro chip en el programador, dentro del cual escribirá la siguiente sección de datos. Como puede saber el estado del conmutador LECTURA/PROGRAMACION, también le lanzará un mensaje para que coloque usted el conmutador en la posición adecuada. El programa puede parecer algo lento, esto se debe a que lee cada octeto que escribe antes de pasar al siguiente, y esto implica cambiar el modo del PPI por cada octeto que se introduce . Se ha preferido este método al de una pasada de lectura una vez escrita toda la EPROM, **porque cualquier fallo hace** que se pare inmediatamente el programa.

El programa se lista en la figura 3.69. Primero imprime sus opciones, parámetros, etc. Después pregunta si quiere usted definir el tipo de EPROM que va a usar. Debe elegir una de las tres posibles respuestas, pulse ENTER solamente si va a usar la 2764, o puede teclear "2764", o "27128". Cualquier otra respuesta será rechazada, y le hará la pregunta de nuevo. El programa le dice después la extensión de la dirección LOCAL para el tipo de EPROM que ha seleccionado, y le pide la dirección de principio de LOCAL para empezar a programar desde ella. Una vez terminados los pasos preliminares, el programa comienza su ejecución estableciendo de dónde debe tomar los datos que va a grabar en la EPROM. Después le preguntará si los datos están en memo-

ria o en un fichero ASCII. Si selecciona memoria como fuente de los datos, le preguntará cuál es la dirección de comienzo (Recuerde que puede introducir valores hexadecimales en una sentencia INPUT mediante el prefijo "&"). Después el programa le preguntará cuál es la última dirección que debe copiar en la EPROM. En este punto puede poner la dirección real de final de datos, o si conoce la longitud de los datos, puede responder "+xx", donde xx es el número de octetos en Hex o en decimal, el programa añade xx a la dirección de comienzo. Si especifica que la fuente de datos es un fichero ASCII, le preguntará por el nombre del fichero e intentará abrir el fichero en cinta o en disco.

La subrutina situada en las líneas 5000 a 5080 es la que hace la trasferencia real de datos a la EPROM. Recibe el dato a trasferir a la EPROM en la variable BLOW. Las rutinas de obtención de datos usan sentencias PEEK e INPUT LINE para obtener un octeto cada vez y pasarlo a esta subrutina en la variable BLOW. Después de programar el octeto, la subrutina lee y comprueba la EPROM para asegurarse de que la trasferencia se ha hecho satisfactoriamente, si ocurre algún error, llama a la subrutina de registro de errores en la línea 6000. Esta imprime los detalles del error - incluido la máscara binaria del octeto que ha escrito y la diferencia con el que ha leído. En este punto puede decidir que termine el programa o que continúe. El listado está suficientemente comentado, de forma que le será fácil modificarlo si lo desea.

```
500  '**          Programa número 1 para programador de EPROM          **
     ** Este programa acompaña al texto del libro "Hardware del CPC"  **
     ** de Alan Trevennor                                             **
900  MEMORY 15000: BBYTES=0
1000 MODE 2: OUTCOME$="correctamente": PRINT STRING$(79,"#"): PRINT TAB(15):
     "Programa de grabación de EPROM - detalles completos en el libro":
     PRINT STRING$(79,"#"): LOCATE 1,5: PRINT "Este rpograma ": PRINT
1010 PRINT "1) Toma datos de memoria, o de un fichero ASCII":
     PRINT "   y los graba en una EPROM.": PRINT: PRINT
     "2) Avisa si hay algún fallo, dando la opción de continuar."
1020 PRINT: PRINT "3) Le permite continuar con otra EPROM     -":
     PRINT "   si los datos no le caben en un solo chip EPROM": PRINT:
     PRINT: PRINT TAB(10);"Notas de uso:"
1030 PRINT: PRINT "Puede usar chips EPROM 2764 (8kx8) ó 27128 (16kx8).":
     PRINT: PRINT "El número de octetos a trasferir incluyen el primero"
1040 PRINT "y el último octeto del rango especificado":
     LOCATE 1,25: PRINT "Pulse cualquier tecla para continuar":
     WHILE INKEY$="": WEND: LOCATE 1,5: PRINT CHR$(20);:
1050 INPUT "Tipo de EPROM que va a programar: 2764 o 27128 <2764>";ETYP:
     IF ETYP <> 27128 AND ETYP <> 2764 AND ETYP <> 0 THEN 1050 ELSE
     LOCATE 1,5: PRINT CHR$(20);: IF ETYP=0 THEN ETYP=2764
1060 PRINT "Al primer octeto de la EPROM le llamamos dirección Local cero."
     PRINT "A la última dirección de la EPROM, la llamamos dirección local";:
     IF ETYP=27128 THEN PRINT 16384 ELSE PRINT 8191: PRINT
1070 INPUT "Dirección local del octeto de la EPROM donde empezar a programar
     <0>"; LOCAL: IF LOCAL <0 OR LOCAL > 16384 THEN PRINT "No seas estúpido!":
     GOTO 1070
1075 IF LOCAL > &1FFF AND ETYP =2764 THEN PRINT
     "Dirección Local ilegal para un 2764 - pruebe de nuevo": GOTO 1070
1080 IF (INP(&F8E4) AND 2) =0 THEN PRINT
     "Coloque el conmutador PROG/LEER en posición PROG": WHILE (INP(&F8E4)
     AND 2) = 0: WEND: PRINT "Gracias"
```

```
1090  OUT &F8E3,&80: OUT &F8E1,(INT(LOCAL/256) AND 255):
      OUT &F8E0,(LOCAL AND 255)
1100  '** Ya se ha hecho la inicialización. Ahora debemos especificar  **
       ** de donde vamos a tomar los datos                             **
1110  '**       Fase de selección de la fuente de los datos           **
1120  LOCATE 1,5: PRINT CHR$(20);
1130  PRINT "Ahora debe especificar dentro de ";STR$(ETYP);" la fuente de los ":
      PRINT "datos que va a grabar. Los datos pueden estar en memoria, o en un":
      PRINT "fichero ASCII - ver el texto."
1140  LOCATE 1,10: PRINT "Tomar los datos desde (M)emeoria p (F)ichero? ";:
      WHILE R$="": R$=INKEY$: WEND: R$=UPPER$(LEFT$(R$,1)): IF R$="M" THEN
      PRINT "Memoria" ELSE IF R$="F" THEM PRINT "Fichero" ELSE R$="": GOTO 1140
1150  SOUND 1,115,25: SOUND 2,100,25: IF R$="M" THEN 1200 ELSE
      LOCATE 1,12: PRINT CHR$(20);: INPUT "Nombre del fichero";F$:
      IF F$="" OR LEN(F$) > 16 THEN 1150 ELSE F$=UPPER$(F$)
1160  PRINT: OPENIN F$: LOCATE 1,14: PRINT "Fichero abierto"+CHR$(20): WHILE
      EOF =0: LINE INPUT #9,R$: FOR I=1 TO LEN(R$): BLOW=ASC(MID$(R$,I,1)):
      GOSUB 5010: NEXT
1165  BLOW=&D: GOSUB 5010: BLOW=&A: GOSUB 5010: BLOW=&FF: GOSUB 5010:
      WEND: GOTO 32760
1170  '** La línea 1160 abre el fichero ASCII (error si el tipo de fichero  **
       ** <> ASCII) le dice al usuario que se ha abierto el fichero, después **
       ** toma cada cadena y pasa cada carácter a la EPROM, + CR/LF          **
1175  '** Cuando se ha grabado todo, se programa un carácter &FF para indicar**
       ** que es el final del texto.                                        **
1200  '** Esta sección toma los datos de memoria y los manda a la subrutina  **
       ** BLOW. Primero pide la dirección de comienzo y la longitud de los   **
       ** datos.

1210  LOCATE 1,12: PRINT CHR$(20);:
      INPUT "Dirección del primer octeto de datos";FIRS:
      INPUT "Dirección del último octeto de datos";LAS$
1215  IF LEFT$(LAS$,1) <> "+" THEN LAS=VAL(LAS$) ELSE
      LAS=FIRS+(VAL(MID$(LAS$,2)))
1220  IF LAS < FIRS THEN 1210 ELSE IF FIRS > 65535 THEN 1210 ELSE IF
      LAS > 65535 THEN 1210 ELSE PRINT "Parámetros aceptados": PRINT
1230  PRINT "Los datos comienzan";TAB(30);"Los datos terminan"; TAB(50);
      "Programando en": PRINT "HEX ";HEX$(FIRS);TAB(30);"HEX ";HEX$(LAS)
1240  FOR I=FIRS TO LAS: BLOW=PEEK(I): LOGATE 50,17: PRINT HEX$(I);:
      GOSUB 5010: NEXT: GOTO 32760
5000  '
       **   Esta es la subrutina BLOW. Introduce el contenido de la   **
       **  variable BLOW dentro de la EPROM y la verifica             **
5010  LOCATE 1,20: PRINT "BV ";HEX$(BLOW);TAB(10);"LA "; HEX$(LOCAL);"   ":
      WHILE (INP(&F8E4) AND 1)<> 0: WEND: OUT &F8E2,(BLOW AND 255):
      WHILE (INP(&F8E4) AND 1)<> 0: WEND
5020  LOCAL=INP(&F8E0)+(256*(INP(&F8E1))): OUT &F8E3,&89 '
      ** Salva el valor de la dirección Local, cambia el modo del PPI para
      ** verificar
5030  OUT &F8E0,(LOCAL AND 255): OUT &F8E1,INT(LOCAL/256): CHK=INP(&F8E2):
      IF CHK <> (BLOW AND 255) THEN GOSUB 6000
5040  OUT &F8E3,&80: LOCAL=LOCAL+1: IF LOCAL > &1FFF AND ETYP=2764 THEN
      5060 ELSE IF LOCAL > &3FFF AND ETYP=27128 THEN 5060 ELSE
      OUT &F8E0,(LOCAL AND 255): OUT &F8E1,INT(LOCAL/256)
5050  BBYTES=BBYTES+1: RETURN
5060  LOCATE 1,18: PRINT CHR$(20)+"EPROM llena, por favor coloque otra";
      ETYP;"en el programador": PRINT "Pulse cualquier tecla para continuar -
      o la tecla $ para terminar": X$="": WHILE X$="": X$=INKEY$: WEND: IF X$="$"
      THEN 5080
```

```
5070 OUT &F8E0,0: OUT &F8E1,0: LOCATE 1,18: PRINT CHR$(20): GOTO 5050:
     LOCAL=0: '** Rearranca con el nuevo chip y la dirección Local a cero **
5080 LOCATE 1,18: PRINT CHR$(20);:INPUT "Está seguro de que quiere abandonar la
     ejecución?";R$: R$=UPPER$(R$,1)): IF R$ <> "S" THEN 5060 ELSE
     OUTCOME$="-Abandonado por requerimiento del usuario": GOTO 32760
6000 LOCATE 1,20: PRINT TAB(10);"** Fallo en la dirección local de EPROM";LOCAL
     ;"**": LOCATE 1,22: PRINT "Detalles: Valor del octeto escrito";
     BLOW;"(";BIN$(BLOW);")": OUTCOME$="" '** Limpia la cadena
6010 PRINT "Valor del octeto leido de la EPROM ";CHK;"(";BIN$(CHK);")":
     PRINT: PRINT "Desea continuar?": X$="": WHILE X$="": X$=INKEY$: WEND:
     X$=UPPER$(LEFT$(X$,1)): IF X$ <> "S" THEN 32760
6020 LOCATE 1,20: PRINT CHR$(20): RETURN
32760 LOCATE 1,5: PRINT CHR$(20);"Coloque el conmutador PROG/LEER en posición
     LEER": WHILE (INP(&F8E4) AND 2) <> 0: WEND: LOCATE 1,5: PRINT CHR$(20);
     "Gracias - ** Ejecución completada ";OUTCOME$;" **"
32765 PRINT CHR$(20);CHR$(24);BBYTES;"octetos grabados -Apúntelo AHORA! ";
     CHR$(24)
32767 END
```

Fig 3.69 Programa para grabar una EPROM con el programador

Notas de uso

El programador de EPROM es un dispositivo muy útil para su CPC. Le permite desarrollar aplicaciones que sin él le resultarían imposibles. Le permitirá, por ejemplo, desarrollar un proyecto que funcione en solitario dentro de su CPC. Estamos fabricando ROMs a medida del usuario - que es lo que realmente representa una EPROM.

Otra aplicación es construir una ROM de conversión de códigos, para su uso en diversos proyectos. Por ejemplo, usted puede crear una configuración 'hardware' en la que puede conectar una EPROM especial para decodificar datos en un equipo de seguridad. La forma en que se puede hacer es introducir un código en las líneas de dirección de la EPROM, y el equivalente codificado aparecerá en las líneas de datos. Un requerimiento muy frecuente es la conversión de ASCII a otro de los muchos códigos internacionales. La figura 3.70 nos muestra un diagrama simplificado de cómo se puede usar una EPROM, programada especialmente para la conversión de códigos.

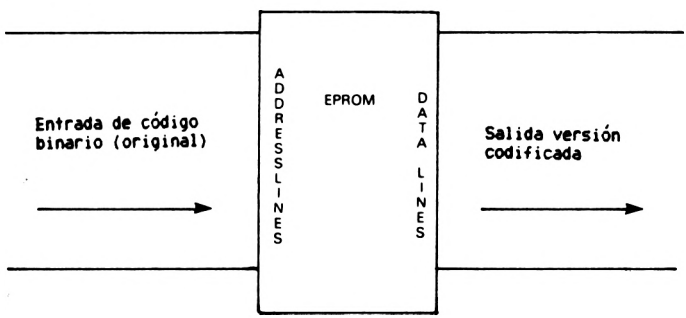


Fig 3.70 Diagrama que muestra a una EPROM como conversor de códigos

Usted puede programar una EPROM con datos para el uso en un sistema experto, donde la EPROM contenga varias preguntas y acciones a las que pueda acceder un programa (que puede estar en otra EPROM) e imprimirlas para el usuario del sistema.

Se puede crear una EPROM que contenga los códigos que se deben introducir en un chip sintetizador, como el AY-8192 del CPC. Si usa un zócalo ZIF tendrá la base para una caja musical electrónica.

Usted puede almacenar en una EPROM información relativamente estática, como una lista de números de teléfono y direcciones. Así lo tendrá siempre disponible instantáneamente para imprimirlo en la pantalla.
¡También puede construir un disco en ROM..... Pero esto es parte del siguiente proyecto!

La lista de componentes del programador de EPROM aparece listada en la figura 3.71, y la tabla de conexión de la alimentación de los chips en la figura 3.72.

Condensadores

C1	22 mfd 10 voltios Radial
C2-C5	0.1 mfd 10 voltios disco

Resistencias

R1	5K6 Ohmios 1/4 watio
R2	270 Ohmios 1/4 watio
VR1	2K Ohmios potenciómetro ajustable

Semiconductores

LED1	LED rojo intermitente
LED2	LED amarillo
E1	7425 Dos puertas NOR de cuatro entradas con sincronismo
E2	74LS20 Dos puertas NAND de cuatro entradas
E3	74LS138 Decodificador de 3 a 8 líneas
E4	74LS02 Cuatro puertas NOR de dos entradas
E5	74LS04 Inversores hexadecimales
E6	8255A Chip PPI
E7	74LS126 cuatro 'buffers' triestado
E8	74LS123 Monoestable doble con disparo

Varios

Zócalos para todos los chips
Zócalo ZIF de 28 patillas
SW1 conmutador de dos circuitos, tres posiciones
con las patillas en ángulo recto.

Circuito impreso
SKT1 Conector pequeño
Conector de expansión de 'bus'

Fig 3.71 Lista de componentes del proyecto de programador de EPROM

Conclusión

Muchos comercios venden borradores de EPROM en cajas. Estas contienen el tipo correcto de lámpara UV, y normalmente un temporizador que enciende la fuente de luz UV durante el tiempo necesario para borrar la EPROM que se introduce en la caja. Hubiera sido interesante describir la forma de construir uno mismo una unidad de este tipo, pero el espacio disponible nos limitaba el número de proyectos que podíamos incluir en este libro. Otro punto a tener en cuenta relacionado con el borrado es que la luz del sol incidiendo directamente sobre la ventana de la EPROM puede borrar parte de ella, por lo tanto no almacene sus chips EPROM cerca de una ventana o en un coche.

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	OV
1	74LS21	2 puertas AND 4 entradas	14	--	--	7
2	74LS20	2 puertas NAND 4 entradas	14	--	--	7
3	74LS138	Decodificador 3 a 8 líneas	16	--	--	8
4	74LS02	4 puertas NOR 2 entradas	14	--	--	7
5	74LS04	Inversor Hex	14	--	--	7
6	8255A	Chip PPI	26	--	--	7
7	--	Zócalo ZIF	28	--	--	14
8	74LS126	4 'buffer' triestado	14	--	--	7
9	74LS123	2 Monoestables	16	--	--	8

Fig 3.72 Patillas de alimentación de los chips del programador de EPROM

PROYECTO 9: DISCO EN ROM

Con la aparición de dispositivos EPROM de gran capacidad como el 27128 de 16K x 8 bits, se ha creado la posibilidad de reemplazar un disco de solo lectura con varias EPROMS. Si pensamos un momento sobre el uso de un ordenador personal, veremos que casi siempre cargamos los mismos programas en memoria una y otra vez. Estos pueden ser ensambladores, juegos, o procesadores de texto. Cargarlos desde el cassette ocupa bastante tiempo, dependiendo directamente de la longitud del programa. Cargarlos desde disco es más rápido, pero es aún más rápido un disco en ROM. Durante el periodo de tiempo en que se usa un disco en ROM, se evitan los problemas que pueden tener los discos - ya que en algunos programas no es necesario ni tener un disco cargado. En relación con el costo, los diskettes son más baratos que los chips EPROM, y se pueden reusar muchas veces, pero el tiempo de iniciación en un disco en ROM es menor que en una unidad de disco (aún a los bajos precios de estas unidades en el Amstrad), por lo tanto puede ser atractivo para algunos lectores este tipo de unidad. Este proyecto es sumamente útil, sobre todo cuando se tienen que almacenar grandes cantidades de información estática.

El concepto de disco en ROM es el de una tarjeta que contiene varias chips EPROM, cada uno de los cuales tiene dentro un programa de los que más se usan. Gracias a la forma modular en la que está escri-

to el 'software' del CPC, es muy fácil introducir comandos nuevos contenidos en una expansión de las rutinas de la ROM que haga que se carguen los programas desde las EPROMs a la memoria.

El diseño del disco en ROM que vamos a ver usa el concepto de 'bus' LOCAL o paralelo descrito en el programador de EPROM. Como necesitamos una forma de programar los chips EPROM con los programas requeridos, obviamente tenemos que construir el programador de EPROM junto con la tarjeta del disco en ROM, y la tarjeta de expansión de la ROM para contener el programa que implementará los nuevos comandos.

Los objetivos de diseño del disco en ROM son los siguientes:

1) Que sea posible acomodar hasta 8 chips EPROM.
2) Expansión para las tarjetas EPROM que pueda construir el usuario.
3) Una ROM de expansión contiene varias rutinas que permiten acceder fácilmente a los ficheros almacenados en el disco en ROM.
4) Usar el mínimo espacio posible en las direcciones de E/S.

El segundo punto se ha incluido porque es fácil imaginar que algunos lectores querrán tener más de ocho programas almacenados en el disco en ROM. Esto significa que se requerirá añadir otra tarjeta EPROM. El diseño que hemos desarrollado le permitirá conectar su propia tarjeta EPROM. Usando este método se pueden añadir hasta un total de 255 chips EPROM. Esto nos da un potencial de hasta 4 megaoctetos (más de cuatro millones de octetos) de almacenamiento, usando 255 chips 27128. La figura 3.73 muestra la disposición en un diagrama de bloques.

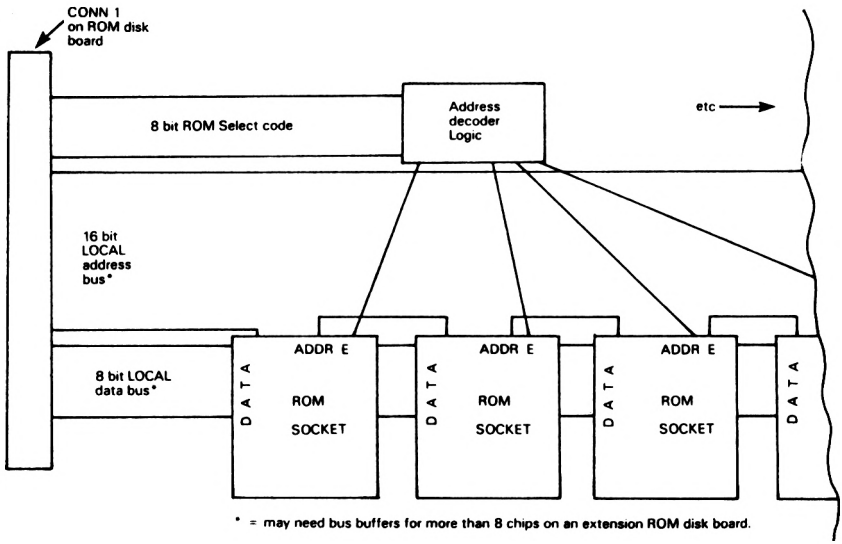


Fig 3.73 Diagrama de bloques que muestra cómo se pueden conectar varias tarjetas extra al disco en ROM

Las EPROM son chips LSI (Large Scale Integration - Integración a Gran Escala) y por lo tanto consumen una cantidad relativamente alta de corriente desde la línea de +5V. Antes de decidir el número de tarjetas con EPROM que va usted a tener, asegúrese de que no va a sobrecargar la línea de +5V de su fuente de alimentación. Refiérase a la tabla de la figura 3.74 que muestra el consumo de cada chip EPROM que vamos a usar, y cuanto va a consumir cada grupo de 4 chips adicionales. (Las figuras publicadas son idénticas tanto para 2764 como para 27128). Estos son los valores cuando el chip es deseleccionado (esto es, su entrada CE está a nivel alto). Cuando se selecciona una EPROM, consume aproximadamente el doble como máximo, pero como el diseño asegura que solamente puede haber una seleccionada a un mismo tiempo, no se debe tener en consideración.

Tabla de consumo de corriente de las EPROM

Chips	AMPS	Chips	AMPS	Chips	AMPS	Chips	AMPS
8	0.4	72	3.6	136	6.8	200	10
12	0.6	76	3.8	140	7	204	10.2
16	0.8	80	4	144	7.2	208	10.4
20	1	84	4.2	148	7.4	212	10.6
24	1.2	88	4.4	152	7.6	216	10.8
28	1.4	92	4.6	156	7.8	220	11
32	1.6	96	4.8	160	8	224	11.2
36	1.8	100	5	164	8.2	228	11.4
40	2	104	5.2	168	8.4	232	11.6
44	2.2	108	5.4	172	8.6	236	11.8
48	2.4	112	5.6	176	8.8	240	12
52	2.6	116	5.8	180	9	244	12.2
56	2.8	120	6	184	9.2	248	12.4
60	3	124	6.2	188	9.4	252	12.6
64	3.2	128	6.4	192	9.6	256	12.8
68	3.4	132	6.6	196	9.8		

Fig 3.74 Tabla de consumo de corriente de 1 a 255 chips EPROM

Necesitamos asignar solamente cinco direcciones de E/S para el disco en ROM. Las que hemos elegido, y sus usos, son las siguientes:

Hex FAE0 = PPI puerta A - usado para los bits 0-7 del 'bus' LOCAL
Hex FAE1 = PPI puerta B - usado para los bits 8-15 del 'bus' LOCAL
Hex FAE2 = PPI puerta C - usado para los bits 16-23 del 'bus' LOCAL
Hex FAE3 = PPI octeto de control - Todas las puertas son de salida.
Hex FAE8 = Registro de lectura de datos, lee el estado del 'bus' de Datos
LOCAL

Como podemos ver en la sección del 'software' del disco en ROM, la primera ROM debe estar instalada siempre, y se usa como directorio del disco en ROM.

Circuito del Disco en ROM

La parte electrónica del disco en ROM se muestra en el diagrama de bloques de la figura 3.75. Esta nos muestra que a las líneas de dirección y datos de las ROMs están conectadas los 'bus' de direcciones y datos generados localmente. El bus de direcciones locales es gene-

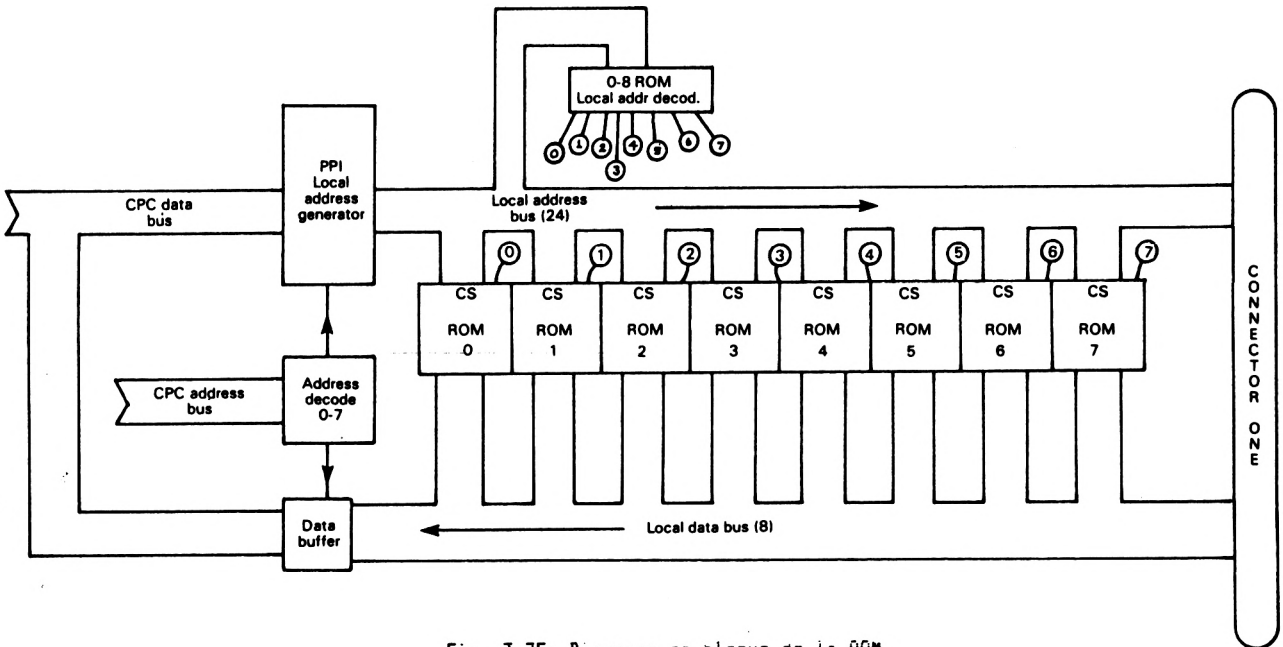


Fig. 3.75: Diagrama de bloque de la ROM.

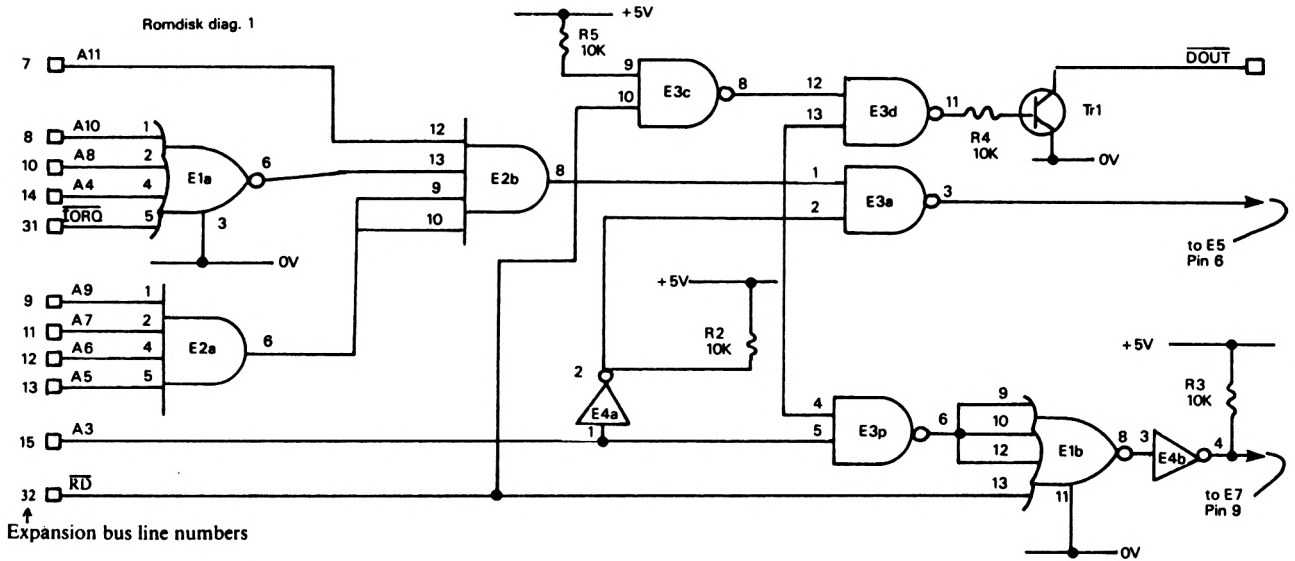


Fig. 3.76: Decodificador de direcciones de disco en ROM.

I	I	I	I	I	O	I	O	I	I	I	O	O	O	O	O
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
X				A				E				O			

X = dont care (set to F)

rado por el PPI, y las líneas de datos de todas las ROM van a parar al bus de datos del CPC, a través de un chip 'buffer' triestado. Además, el 'bus' local de datos y el de direcciones se prolongan fuera del conector, con lo que nos permitirá construir las extensiones del disco en ROM que necesitemos. La figura 3.76 es la primera sección del circuito. Nos muestra el decodificador de direcciones. Este decodificador en particular se ha hecho todo con puertas lógicas. Los estados de los bits, requeridos para formar los tres dígitos inferiores de las direcciones, se muestran recuadrados en la parte inferior de la figura. El efecto concreto de todas estas puertas es que la patilla 3 de la puerta E3a se ponga en estado lógico cero cuando sea direccionado cualquiera de los registros del PPI, y la patilla 4 del inversor E4b se ponga en estado lógico cero cuando se direccione un registro de datos separado. La función de las puertas que hay alrededor del transistor TR1 es detectar cuándo se han leído datos del disco en ROM, y hacer conducir al transistor, poniendo la señal DOUT del 'bus' de expansión, a nivel activo bajo. (Ver apéndice cinco para la descripción de las señales del 'bus' de expansión, incluida DOUT). En este caso, E4 no es un chip inversor 74LS04, sino una versión del mismo, pero de colector abierto, por lo que debemos mantener altas sus salidas mediante resistencias.

Vayamos ahora a la figura 3.77, que nos muestra el chip PPI y parte del conector de expansión - CONN uno (que veremos más adelante). La señal de selección se manda desde el decodificador de direcciones y está conectada a la entrada a la entrada CS negativa del 8255A. Las señales WR negativa y RD negativa del 'bus' de expansión, se introducen directamente en las entradas correspondientes del PPI. La señal RESET del 'bus' de expansión, se invierte mediante E4c antes de ir a parar a la entrada RESET del PPI. Esto se hace así porque el PPI necesita una señal de restauración alta. Las líneas de datos del PPI se conectan simplemente a las líneas del 'bus' de expansión, y las entradas A0 y A1 del PPI se conectan a las líneas de dirección A0 y A1 del 'bus' de expansión. Esto significa que, una vez que el decodificador de direcciones ha decodificado una dirección que comienza por AE, y el bit 3 de dirección está a cero, el registro interno del chip PPI que se selecciona depende enteramente del estado de las líneas de dirección, A0 y A1. Como ya dijimos anteriormente, las tres puertas, A, B y C, se usan en orden ascendente, para generar una dirección local de 24 bits. Las líneas de dirección local se conectan también a CONN1, de forma que se puedan hacer fácilmente las extensiones al disco en ROM.

El siguiente esquema corresponde al decodificador de direcciones locales, y se muestra en la figura 3.78. Debido a que los chips EPROM que estamos usando, tienen solamente doce o trece líneas de dirección, tenemos que decodificar los ocho bits superiores de la dirección local, para producir las señales de selección de chip que se pasan a la EPROM requerida. Los inversores E4d-E4f, se conectan juntos en una configuración OR, de forma que si cualquiera de ellas tiene una salida baja, la salida de las otras se pondrá también baja. Si las entradas de estos inversores es baja, las salidas se ponen todas altas y la resistencia R1 fuerza la entrada G1 de E6 a un nivel alto. Las líneas de dirección locales, LA16-LA18, se introducen en las entradas de selección de A, b y C del decodificador, junto con LA19 y LA20 (introducidas en las entradas G2 del decodificador) que se ponen a nivel bajo, la salida Y correspondiente pasará a nivel bajo, para seleccionar uno de los ocho chips EPROM de la tarjeta.

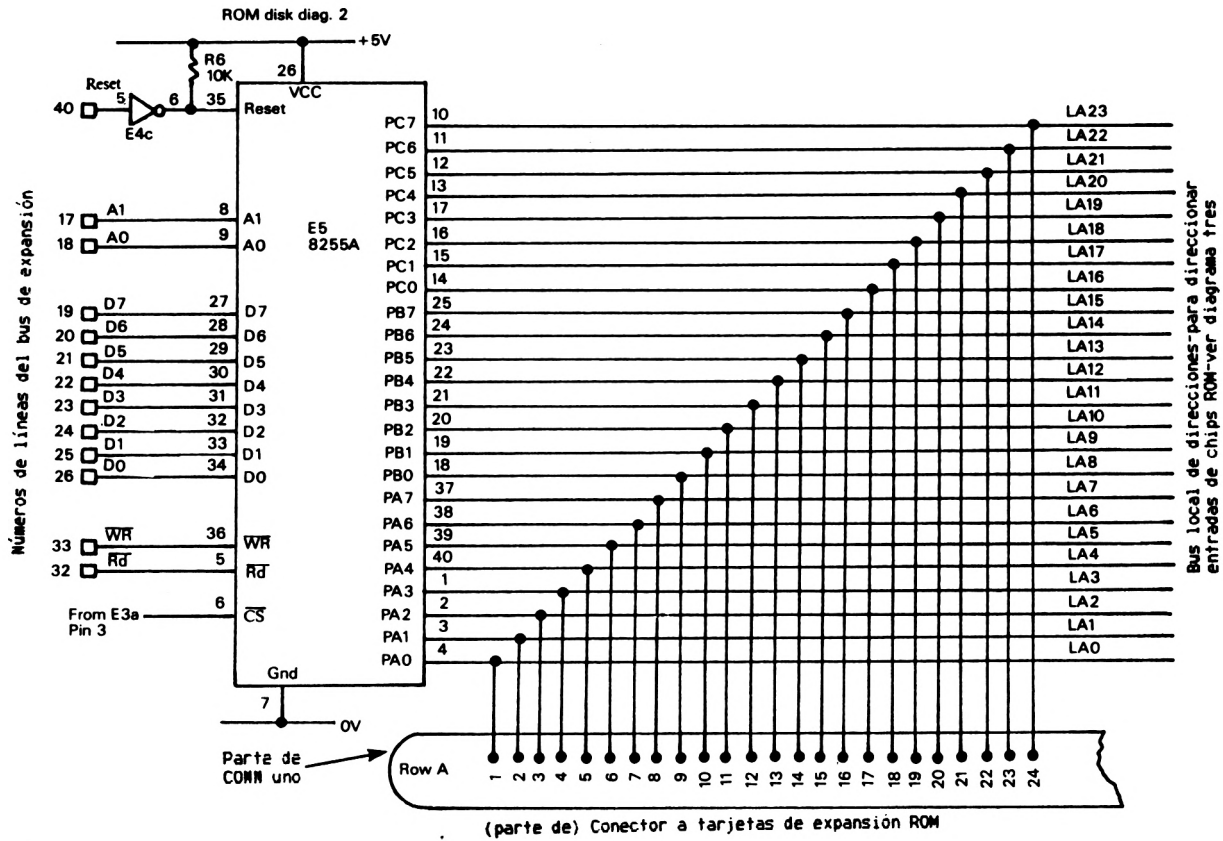


Fig 3.77 Chip PPI y conector de expansión del disco en ROM

El esquema final del 'hardware' del disco en ROM, se ve en la figura 3.79. Esta figura nos muestra el 'buffer' del 'bus' local de datos, chip E7, y la conexión típica de un zócalo para EPROM. Para empezar con el 'buffer' del 'bus', cuando el Z80A selecciona las direcciones de E/S xAE0-xAE3, se activa E7, poniendo la entrada G negativa a nivel bajo, para pasar el contenido del 'bus' local de datos. Esto sucede cuando la lógica del decodificador de direcciones detecta una lectura del disco en ROM. Los zócalos de la EPROM están conectados en paralelo, excepto sus entradas CE negativas. La patilla 20 de cada EPROM se conecta al decodificador local de direcciones. Esto significa que, como el decodificador local de direcciones es capaz de de sacar una sola señal en un momento determinado, solamente se puede seleccionar una EPROM en ese instante. El 'bus' local de datos continúa también hasta CONN1, de forma que se pueden añadir tarjetas EPROM extra, cuando los ocho zócalos de la tarjeta del disco en ROM se hayan completado. Los condensadores C1-C5 se usan para desacoplar y filtrar las líneas de alimentación del circuito.

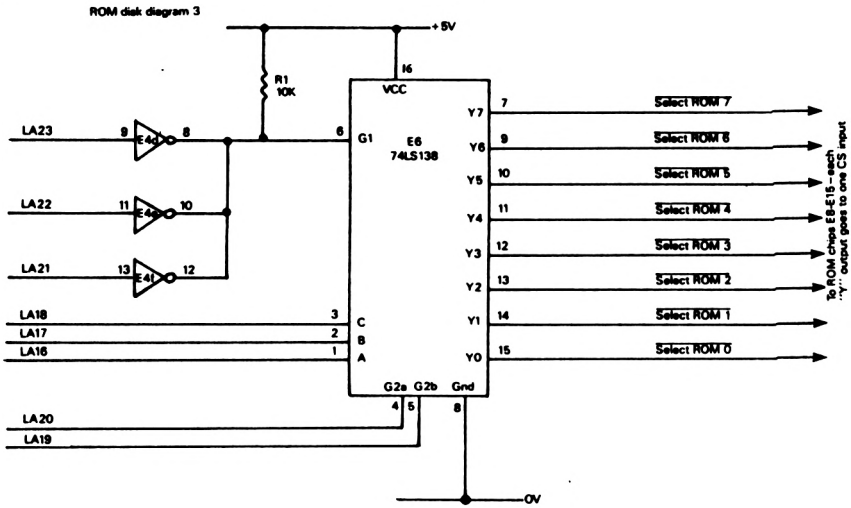


Fig 3.78 Decodificador de direcciones locales del disco en ROM

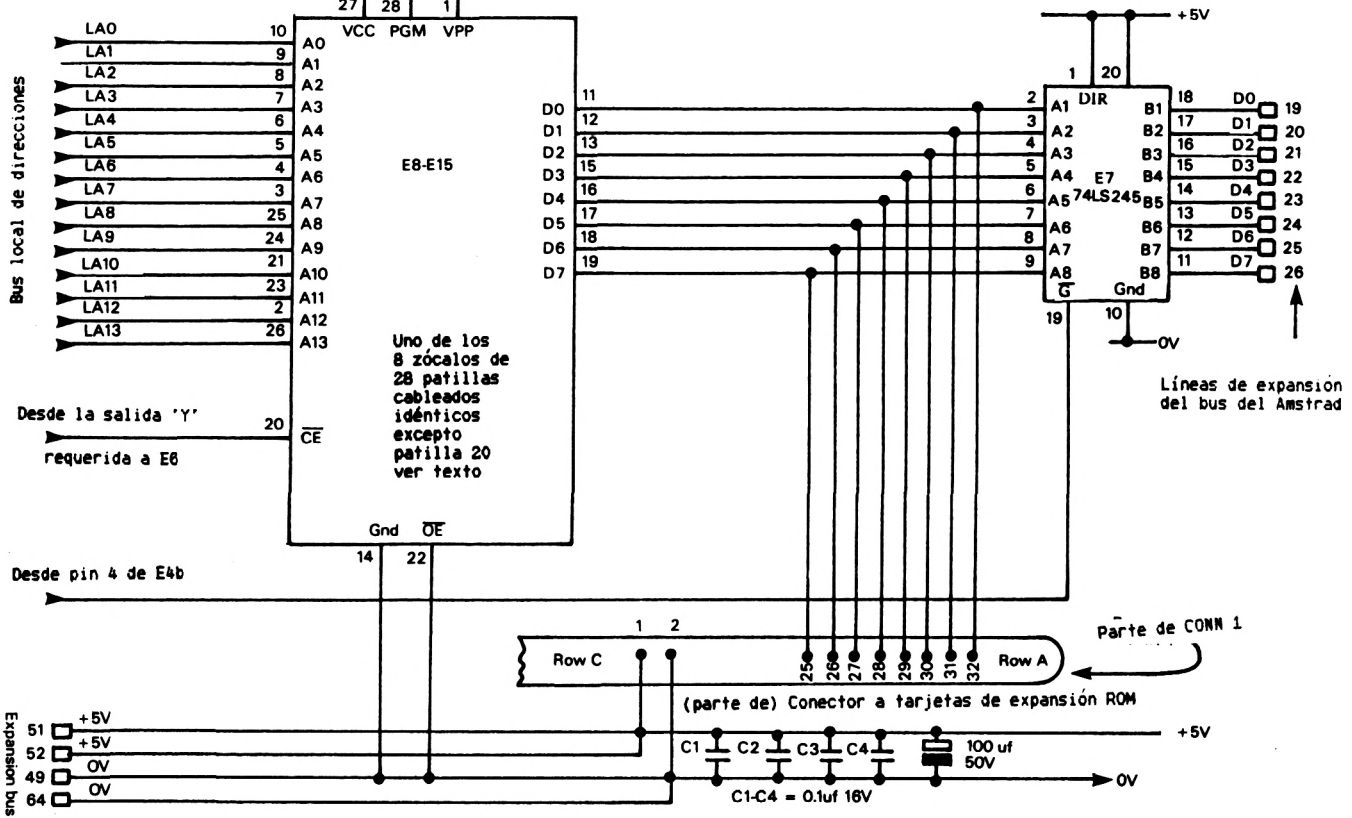
Requerimientos para la tarjeta de extensión del disco en ROM

Como ya mencionamos anteriormente, el conector CONN1 sirve para conectar sus propias tarjetas de expansión del disco en ROM. Se usará cuando los ocho zócalos de la tarjeta principal no sean suficientes para contener todo lo que queremos almacenar en el disco en ROM.

Las tarjetas de extensión del disco en ROM solo necesitan contener el decodificador local de direcciones, para seleccionar el número de ROM requerida (Para la primera serán los números que van a continuación de los de la tarjeta principal, de forma que la primera ROM de la tarjeta de extensión será la ROM 8). Para permitir añadir futuras tarjetas de expansión, después de la primera, cada una de ellas debe tener un equivalente de CONN1, de forma que tengan continuación los 'bus' locales de dirección y datos.

Diag. 4

Fig 3.79 Zocalo típico de ROM y 'buffer' del 'bus' del disco en ROM



'Software' del disco en ROM

El 'software' del disco en ROM le será bastante difícil de entender si no conoce bien la ROM del CPC. Se explica detalladamente en un libro de Amsoft denominado SOFT 158. Si usted no tiene acceso a los detalles de la ROM, podrá cargar y ejecutar estos programas, aunque tendrá dificultad en comprenderlos en detalle.

Debido a que los programas del disco en ROM están íntimamente relacionados con la ROM del CPC, la descripción detallada asumirá que usted tiene acceso a SOFT 158. La descripción general será suficiente para darle a los lectores detalles de los comandos y opciones que proporciona la adición del 'software' del disco en ROM.

Descripción general

Los programas del disco en ROM están contenidos en lo que se conoce como una ROM secundaria. Esta es una EPROM que contiene los programas necesarios para que funcione el disco en ROM, y que proporciona varios comandos extra que pueden incluirse en programas o introducirse a través del teclado. Esta ROM secundaria se instala en la tarjeta de expansión de ROM, que forma nuestro siguiente proyecto. En términos generales, cada ROM secundaria se puede conmutar, bajo control de la ROM del CPC, dentro del espacio de direcciones que ocupa normalmente el BASIC, esto es, desde Hex COOO a Hex FFFF. A pesar de ocupar el mismo espacio de direcciones, el BASIC y los programas del disco en ROM se pueden comunicar entre sí por medio de las rutinas que hay en la parte inferior de la ROM. Como en todos los comandos extra, la barra vertical (@ en mayúsculas) debe precederlos. Los nuevos comandos se explican junto a la descripción de las facilidades que proporcionan.

El disco en ROM proporciona un medio de almacenamiento extra para ficheros binarios y ASCII, en una máquina CPC. En términos prácticos esto significa que puede cargar programas BASIC que han sido introducidos en una EPROM (usando el programador de EPROM). Puede cargar programas en código máquina (como el DEVPAC) o sus propios datos desde el disco en ROM. Veamos cada una de estas opciones.

El CPC BASIC tiene la capacidad de cargar programas almacenados en formato ASCII y convertirlos en su propio formato interno, después de lo cual pueden ser ejecutados. Cuando usted salva un programa con el formato - SAVE "PROGRAMA",A - hace que se salve en disco o en cinta la versión ASCII de su programa. Lo que hacen los programas del disco en ROM es interceptar las peticiones del BASIC para cargar ficheros, haciendo que cuando usted teclee los comandos LOAD, RUN, MERGE u OPENIN, se carguen los programas salvados en forma de texto desde, el disco en ROM al BASIC. En definitiva, los programas del disco en ROM reemplazan ciertas rutinas de la ROM del CPC, de forma que el BASIC no se da cuenta de la diferencia. Esta clase de desviación solo es posible gracias a la política abierta que ha seguido Amstrad en el 'software'. Para crear esta desviación debe usar el comando "|RD.OPEN" que 'abre' el disco en ROM al BASIC. Cuando quiera volver atrás para cargar desde el disco o la cinta, teclee el comando "|RD.CLOSE" que elimina la desviación.

La carga de código máquina (o datos binarios) se realiza con el comando "|RD.LOAD,R,A". Donde R es el número de ROM del que tiene que

cargar, y A es la dirección en la que debe comenzar a cargar. Así, por ejemplo, el comando "|RD.LOAD,5,1000" cargará la ROM 5 en memoria empezando en la dirección 1000. Como la ROM cero contiene el directorio del disco en ROM, el 'software' de RD.LOAD sabe cuantos octetos están usados en cada chip EPROM y carga esos octetos en la memoria del CPC. RD.LOAD le dirá si tiene suficiente memoria para cargar la ROM seleccionada. Si no se ha creado la entrada del directorio del zócalo EPROM, no le será posible usar RD.LOAD para cargar su contenido en memoria, ya que RD.LOAD le dará el mensaje de error "** Esta ROM no está grabada **". Lo mismo sucede si especifica en el comando un número ilegal de ROM.

Puede listar lo que tiene cargado en el disco en ROM mediante el uso del último de los nuevos comandos asociados con el 'software' del disco en ROM. Este último comando es "|RD.DIR", que muestra en la pantalla del CPC una línea con el contenido de cada una de las entradas que hay en la ROM de directorio - conectada en el zócalo cero. Si necesita imprimir el directorio, debe usar el comando "|RD.DIR,P" que no solo muestra el directorio en la pantalla del CPC, sino que saca también una copia por la puerta de la impresora. (vea la discusión del directorio en ROM más adelante). Con la información que obtenemos por medio de este comando, podemos decir qué ROM hay que usar para cargar un fichero determinado. Esto resulta muy útil, ya que no se puede acceder por nombre de fichero, solamente por el número de ROM donde reside.

Directorio del disco en ROM

El tipo usual de información que usted almacenará en el disco en ROM, será del tipo que no necesita modificarse o borrarse. Cuando quiera añadir una nueva ROM, deberá actualizar el contenido de la ROM cero, que es el directorio de todas. Ya que la necesidad de actualizar la ROM directorio no será frecuente, podrá usar un zócalo DIL estándar para la ROM cero. El introducir y quitar la ROM cero de este zócalo no debe causar serios problemas, pero si lo va a hacer muchas veces, será mejor que instale un zócalo ZIF. Si piensa actualizar regularmente el contenido de su disco en ROM, puede hacerlo, pero debe tener en cuenta los siguientes puntos:

1) Use un zócalo ZIF para la ROM 0 de la tarjeta de disco en ROM
2) Asegúrese de que tiene salvado el contenido de la ROM cero en un fichero binario en disco o cinta.
3) Use el programa listado en la figura 3.80 para añadir nuevas entradas. Necesitará un borrador de EPROM. (vea el proyecto anterior).

Si tiene en cuenta estos puntos, no tendrá problemas con las actualizaciones regulares del contenido del disco en ROM. La cosa más importante a recordar siempre es que el directorio en ROM debe ser siempre exacto. Esta es su responsabilidad, debe actualizarlo siempre que añada una nueva EPROM.

```
100  ' ** Programa de actualización de la ROM directorio del disco en ROM **

150  '** Este programa añade nuevas entradas a la ROM directorio en la
     ** tarjeta de disco en ROM. La ROM directorio debe ser desmontada
     ** de la tarjeta y montada en el programador de EPROM
```

```
200   MODE 2: PRINT STRING$(79,&2A): PRINT TAB(15):
      "Programa de actualización del directorio del disco en ROM":
      PRINT STRING$(79,&2A)
250   LOCAL=0: GOSUB 10000: IF RED <> 0 THEN 300 ELSE LOCATE 1,5:
      PRINT "Coloque la EPROM directorio en el programador y pulse ENTER":
      WHILE R$ <> CHR$(13): R$=INKEY$: WEND: GOTO 250
300   LOCATE 1,5: PRINT CHR$(20): LOCAL=&20: ENTRY=1 '
      ** Limpia la pantalla y pone LOCAL = primera dirección de entrada local
      ** y ENTRY apunta a la entrada uno (La cero está reservada)
400   GOSUB 10000: IF RED <> &21 THEN 500 ELSE ENTRY=ENTRY+1:
      LOCAL=ENTRY*32: IF ENTRY < 256 THEN 400 ELSE PRINT
      "Esta ROM directorio está llena !!": GOTO 32767
500   PRINT "Crear una entrada nueva para el zócalo ROM";ENTRY;"? <Si> ";:
      INPUT R$: IF R$="" OR UPPER$(LEFT$(R$,1)) ="S" THEN 600
510   INPUT "Qué número de zócalo (1 a 255)";ENTRY :
      IF ENTRY >1 AND ENTRY <256 THEN 520 ELSE LOCATE 1,5:
      PRINT CHR$(20);"Fuera de rango - pruebe de nuevo": GOTO 510
520   LOCAL=(ENTRY*32): GOSUB 10000: IF RED <> &21 THEN 600 ELSE
      LOCATE 1,5: PRINT CHR$(20);"Esa entrada ya está usada":
      GOTO 510
600   '** Cuando llega aquí, ENTRY contiene el número de la entrada que va
       ** a crear, y LOCAL contiene la dirección local de su primer octeto.
610   LOCATE 1,5: PRINT CHR$(20);"Entrada de descripción del contenido de la ROM:"
620   LOCATE 1,7: PRINT CHR$(18);"Nombre de fichero de los datos en ROM":ENTRY;:
      INPUT FIL$: IF LEN(FIL$) > 0 AND LEN(FIL$) < 17 THEN 630 ELSE
      LOCATE 1,22: PRINT "El nombre de fichero debe ser de 1 a 16 caracteres
      imprimibles"
625   GOTO 620
630   LOCATE 1,7: PRINT CHR$(20);CHR$(13);:
      FIL$=FIL$+SPACE$(16-LEN(FIL$)): IF A <> 0 THEN 800
640   PRINT "Dirección de carga";CHR$(18);: INPUT PLADD:
      IF PLADD > 0 AND PLADD < 65536 THEN 650 ELSE LOCATE 1,22:
      PRINT "Esta no es una dirección de memoria": LOCATE 1,7: GOTO 640
650   LOCATE 1,7: PRINT CHR$(20); "Ahora debe especificar el tipo de fichero:":
      RESTORE: PRINT: PRINT "Teclee uno de:": PRINT: FOR I=1 TO 5:
      READ T$(I): PRINT TAB(10);T$(I): NEXT: INPUT "Tipo de fichero";TYP$
660   IF LEN(TYP$) < 3 THEN 650 ELSE TYP$=UPPER$(TYP$): FOR I=1 TO 5:
      IF TYP$ = UPPER$(LEFT$(T$(I),LEN(TYP$))) THEN 670 ELSE NEXT I:
      PRINT CHR$(7): GOTO 650
670   TYP=(I-1)
680   LOCATE 1,7: PRINT CHR$(20);"Ahora introduzca la longitud del fichero en
      octetos. Si no la sabe": PRINT "pulse simplemente la tecla ENTER":
      INPUT LENGTH: IF LENGTH > 0 THEN 800 ELSE IF LENGTH < 0 THEN 680
690   '** Esta rutina intenta averiguar la cantidad de datos que hay en
       ** la ROM cuyo número está en ENTRY. Si la ROM no está aún montada
       ** en el zócalo, o el disco en ROM está ausente, toma como valor por
700   '** defecto un 27128 completo - esto es 16384 octetos como contador
       ** octetos para el contenido. El programa grabador de EPROM le dirá
       ** cuantos octetos ha grabado, así que tome nota del número!!"
710   OUT &FAE3,&80: OUT &FAE2,ENTRY: IF INP(&FAE8) <> 255 THEN 720 ELSE
      INPUT "El programa no puede determinar el tamaño de la ROM, toma 16384?";
      R$: LENGTH =16384: IF UPPER$(LEFT$(R$,1)) ="S" THEN 800
715   LENGTH= 8192 : INPUT "Toma como defecto 8192?";R$:
      IF UPPER$(LEFT$(R$,1)) ="S" THEN 800 ELSE GOTO 680
720   '** Cuando llegamos aquí ya sabemos el tipo de ROM instalada. ahora haremos
       ** una búsqueda de arriba a abajo de la ROM, para encontrar FF's
725   PRINT "ejecutándose cálculo del tamaño ": TOP=&1F '** Ponemos la dirección
      alta a &1FFF para un 2764. Si la dirección superior está usada, la ponemos
      como un 27128: NOTA: si la EPROM se llena completamente fallará esta rutina
```

```
730   FOR H=TOP TO 0 STEP -1: OUT &FAE1,H: FOR I=255 TO 0 STEP -1:
      OUT &FAE0,I: IF INP(&FAE8) <> 255 THEN 740 ELSE NEXT: NEXT
740   IF TOP <> &3F THEN 750 ELSE TOP=&1F: GOTO 730
750   LOCATE 1,7: PRINT CHR$(20);"El tamaño de datos parece ser";(H*256)+I;:
      INPUT "octetos. Acepta esta estimación?";R$: R$=UPPER$(LEFT$(R$,1)):
      IF R$="" THEN 750 ELSE IF R$="S" THEN 760 ELSE 680
760   LENGTH=(H*256)+I
800   '** Ya hemos introducido toda la información requerida. Ahora vamos
       ** a imprimirla en la pantalla para verificarla.
810   LOCATE 1,7: PRINT CHR$(20);: PRINT "Detalles introducidos": PRINT
      "1) ENTRY describe la ROM del zócalo";ENTRY: PRINT
      "2) El contenido de la ROM tiene el nombre de fichero";FIL$
820   PRINT "3) Dirección de carga elegida";PLADD: PRINT
      "4) Tipo de fichero";T$(TYP+1): PRINT
      "5) Longitud de los datos en Hex ";HEX$(LENGTH);" -Dec";LENGTH;CHR$(10)
830   PRINT "Pulse ENTER para aceptar los detalles u otra tecla para
      reintroducirlos":
      R$="": WHILE R$="": R$=INKEY$: WEND: IF R$=CHR$(13) THEN 850 ELSE
      CLEAR: GOTO 200
850   '** Ya estamos preparados para grabar la información en la ROM
860   LOCATE 1,4: PRINT CHR$(20): PRINT "Fase de programación de la EPROM:":
      IF (INP(&F8E4) AND 2) =0 THEN PRINT
      "Ponga el conmutador PROG/LEER del programador de EPROM en posición PROG"
870   WHILE (INP(&F8E4) AND 2) =0: WEND: PRINT "Gracias":
      OUT &F9E3,&80 '** PPI inicializado y en modo PROG
880   BLOW= ASC("!"): GOSUB 11000: FOR XX=1 TO 16: BLOW=ASC(MID$(FIL$,XX,1)):
      GOSUB 11000: NEXT: BLOW=(PLADD AND 255): GOSUB 11000:
      BLOW=(INT((PLADD/256)) AND 255): GOSUB 11000: BLOW=TYP: GOSUB 11000
890   BLOW=(LENGTH AND 255): GOSUB 11000: BLOW=(INT((LENGTH/256) AND 255)):
      GOSUB 11000
900   LOCATE 1,5: PRINT CHR$(20);"Poner el conmutador en LEER":
      WHILE (INP(&F8E4) AND 2) <> 0: WEND: PRINT "Gracias": GOTO 32767

10000 '** Subrutina para cargar un octeto desde la dirección local de la
       ** ROM en el programador de EPROM (normalmente la ROM directorio)
10010 OUT &F8E3,&89: OUT &F8E0,(LOCAL AND 255):
      OUT &F8E1,(INT(LOCAL/256) AND 255: RED=INP($F8E2): RETURN
11000 '
       **   Esta es la subrutina que graba en la EPROM. El contenido de    **
       ** la variable BLOW se graba dentro de la EPROM y se verifica       **
11010 WHILE (INP(&F8E4) AND 1) <> 0: WEND: WHILE (INP(&F8E4 AND 1) <> 0: WEND
11015 OUT &F8E3,&80: OUT &F8E0,(LOCAL AND 255):
      OUT &F8E1,INT(LOCAL/256)
11017 WHILE (INP(&F8E4) AND 1) <> 0: WEND : OUT &F8E2,BLOW:
      WHILE (INP(&F8E4) AND 1) <> 0: WEND
11020 OUT &F8E3,&89 '** Cambia el modo del PPI para verificar
11030 OUT &F8E0,(LOCAL AND 255): OUT &F8E1,INT(LOCAL/256): CHK=INP(&F8E2):
      IF CHK <> (BLOW AND 255) THEN 11100 ELSE LOCAL = LOCAL+1:
      GOTO 11200
11100 PRINT "** Fallo en verificación **";CHR$(7): PRINT "Detalles:": PRINT
      TAB(10);"Dirección LOCAL =";LOCAL;": Escrito";BLOW;": Leido";CHK: PRINT
      INPUT "Reintento este octeto?";R$: IF UPPER$(LEFT$(R$,1))="S" THEN 11000
11110 LOCATE 1,3: PRINT CHR$(20);"Ejecución interrumpida": GOTO 900
11200 RETURN
32000 DATA "BASIC","Binario","Vuelco pantalla","ASCII","Desconocido"
32767 END
```

Fig 3.80 Programa de actualización del directorio del disco en ROM

No resulta difícil almacenar en el disco en ROM los datos para trabajar con programas BASIC. Todo lo que necesita es bajar HIMEM usando una sentencia MEMORY, y después cargar toda o parte del contenido de la ROM en memoria, mediante un comando "|RD.LOAD,R,A" incorporado en el programa. Los datos cargados se pueden leer en cadenas mediante un bucle FOR-NEXT como este:

```
1000 FOR XX=25000 TO 25200: A$=A$+CHR$(PEEK(XX)): IF PEEK(XX)=13 THEN
     2000 ELSE NEXT XX
2000 REM En este punto tenemos una cadena leida dentro de A$
```

Tenga en cuenta que, en este caso, los datos deben ser texto, si no tienen el retorno de carro o el salto de línea, se puede sobrepasar fácilmente la longitud máxima de la cadena. Con texto no debe haber problemas. Estoy seguro de que los lectores verán rápidamente como se puede extender esta idea a otros métodos de entrada de datos desde el disco ROM al BASIC.

Limitaciones

Como el Locomotive BASIC se ha tratado aquí como una caja negra cuando nos conectamos con él, no se ha probado a emular todos las funciones OPEN y CLOSE. Esto se hace mediante el acceso a las tablas de descripción de canales del BASIC. Por esta razón puede que haya ocasiones en que encuentre problemas al usar la sentencia OPENIN del BASIC, porque se puede sobrepasar el fin de fichero, y el BASIC puede pensar que el canal nueve (#9) ya está abierto. Para solventar el segundo problema es un buen truco hacer CLOSEIN antes de cada OPENIN. Si se hace una nueva versión del 'software' del disco en ROM, esperamos arreglar estos pequeños problemas, pero estos no impiden el uso del disco en ROM en aplicaciones serias.

El 'software' del disco en ROM está disponible, junto con el resto del 'software' presentado en este libro. (ver página tres para los detalles). Al final de la descripción detallada, se proporciona el listado en ensamblador de las distintas rutinas que comprenden el 'software', así como un vuelco en hexadecimal de la ROM paralela en la que reside el 'software', de forma que los lectores que no tienen ensamblador, ni dinero para comprar el 'software', puedan introducirlo como código máquina si lo desean.

Descripción detallada del 'software' del disco en ROM

*** NOTA: Esta sección asume que tiene acceso, o conocimientos del manual de la ROM de su CPC***

El 'software' del disco en ROM, como ya se dijo anteriormente, vive en una ROM paralela (realmente una EPROM montada dentro del zócalo ó en la tarjeta de expansión de la ROM - vea el siguiente proyecto). Al encender la máquina, el CPC mira a ver si hay alguna ROM paralela y la coloca en su lista de posibles gestores de comandos. Si se teclea, o se encuentra en un programa BASIC, un comando que no es válido, se busca en las tablas de comandos de las ROM paralelas. La primera entrada de la tabla que hay al principio de la ROM, corresponde a la rutina de iniciación, a la que llama la ROM principal cuando se enciende la máquina. El resto de las entradas de la tabla forman el bloque de saltos de los comandos propios de esa ROM. Así, la primera

entrada del bloque de salto de la ROM paralela del disco en ROM apunta a la rutina que inicializará las condiciones de operación del disco en ROM.

La función principal de la rutina de inicialización del disco en ROM es preparar ciertas posiciones de memoria para su uso exclusivo. La ROM del CPC le pasará posteriormente la dirección base de este área de memoria en el registro IY, cada vez que llame a una rutina del disco en ROM. En resumen, esta facilidad de la ROM del CPC funciona de la siguiente forma:

1) Durante sus rutinas de inicialización, la ROM detecta si está presente alguna ROM paralela .

2) La ROM llama a la rutina a la que apunta la primera entrada en el bloque de saltos. Esta rutina recibe, en el registro IY, la dirección del límite superior del bloque de memoria libre (en BASIC se llama HIMEM).

3) La rutina del disco en ROM (u otro 'software de ROM paralela - todos hacen lo mismo), realizan la inicialización, y baja la dirección en IY el número de octetos que desea reservar. Después la rutina retorna a la ROM del CPC apuntando ahora al nuevo límite superior de memoria. Ahora, cuando la ROM llame a una rutina de la ROM paralela, pasará esta dirección base del área de trabajo del disco en ROM, a la rutina a la que llama. Esta disposición permite flexibilidad, e impone una disciplina, cuando se escriben rutinas para una ROM paralela, ya que todas las tablas se colocan en memoria en una dirección reubicable, normalmente relativa al valor de IY a la entrada.

El módulo de inicialización del disco en ROM - RDINIT - se lista en la figura 3.81. Esta figura nos muestra la rutina, junto con las sentencias que crean la cabecera de la tabla de la ROM, que espera ver la ROM del CPC en una ROM paralela. RDINIT pone también el modo de operación del chip PPI del disco en ROM. Está programado con un octeto de modo Hex 80, de forma que las puertas A, B y C son de salida. Como ya vimos en el 'software' para el programador de EPROM, siempre que el registro de modo del chip PPI se carga, se limpian todos los enclavamientos de salida. Esto significa que RDINIT no tiene que limpiar la dirección de salida a cero, ya que lo hace el 'hardware' cuando se programa el octeto de modo. Hay otros dos trabajos que realiza RDINIT. El primero es asignarse a sí misma Hex 100 octetos de memoria como área de trabajo, como vimos anteriormente. El segundo trabajo es poner a cero el registro de selección de la ROM del disco en ROM, de forma que después de que se haya ejecutado RDINIT la dirección local está a cero, y se selecciona la ROM cero. Finalmente RDINIT imprime un mensaje de inicialización.

```
100           ORG  #C000
120  ;
130  ;  ** Software del disco en ROM
140  ;
150  ADDRLO:   EQU  #FAE0  ; Primero define las direcciones alta y baja
160  ADDRHI:   EQU  #FAE1  ; del registro del disco en ROM
170  ROMSEL:   EQU  #FAE2  ; Después el registro de selección de ROM
180  CTRLP:    EQU  #FAE3  ; Y la puerta de control del PPI
```

```
190 DATAP:    EQU  #FAE8  ; Y la puerta de datos del disco en ROM
200 ;
210 ROMNUM:   EQU  6      ; Define la dirección de la ROM donde reside en la
220           DEFB #1     ; ROM PARALELA
230           DEFB #1,#1,#0     ; Marca 1 Vers 1 Rev 0
240           DEFW NAMTAB       ; Prepara el apuntador a la tabla de nombres
250           JP   RDINIT       ; Rutina de iniciación de encendido
260           JP   RDOPEN
270           JP   RDCLOS
280           JP   RDDIR
290           JP   RDLOAD
300 NAMTAB:   DEFM "RD INI"     ; Entrada de iniciación
310           DEFB "T"+#80
320           DEFM "RD.OPE"
330           DEFB "N"+#80
340           DEFM "RD.CLOS"
350           DEFB "E"+#80
360           DEFM "RD.DI"
370           DEFB "R"+80
380           DEFM "RD.LOA"
390           DEFB "D"+80
400           DEFB 0            ; Marca del final de la tabla de nombres
410 RDINIT:   DEC  H            ; HL es el límite superior de memoria
420           LD   BC,CTRLP
430           LD   A,#80        ; Poner las puertas A, B y C como de salida
440           OUT  (C),A        ; Escribir el modo en el PPI del disco en ROM
450           DEC  BC           ; Ahora BC apunta al registro de selección de ROM
460           LD   A,#0         ; Preseleccionar ROM cero
470           OUT  (C),A        ; poner el registro a cero
480           LD   IX,INIMES    ; Hace parpadear el mensaje
490           CALL OUTPUT
500           JR   INIDUN       ; y sale
510 INIMER:   DEFB #0A,#0D
520           DEFM "** ROMdisk v1.2 iniciado **   "
530           DEFB #0A,#0D,#0A,#FF
540 INIDUN:   SCF               ; pone el indicador de acarreo para señalar OK
550           RET               ; Fin de RDINIT
```

Fig 3.81 Listado de RDINIT

El área de trabajo de 256 octetos asignada por RDINIT se usa para mantener salvadas las entradas del bloque de saltos, resultados intermedios, apuntadores a las memorias intermedias, etc. En la figura 3.82 se lista el mapa completo. Los octetos del área de trabajo no están en direcciones específicas, ya que los valores variarán de acuerdo con el 'hardware' que tenga su máquina, versión del BASIC, etc. Consecuentemente todas las posiciones del área de trabajo hacen referencia al valor de IY que, como ya dijimos, cuando entra en las rutinas del disco en ROM apunta siempre a la dirección base del área de trabajo. La figura 3.83 muestra otra información del disco en ROM, que puede serle útil como referencia cuando se introduzca en el 'software' del disco en ROM, o cuando intente hacer modificaciones.

El siguiente módulo del disco en ROM se llama RDLOAD. Este aparece listado en la figura 3.84. El propósito de esta rutina es cargar el contenido de la ROM especificada en el comando, dentro de la memoria del CPC, comenzando en la dirección especificada en el comando.

```
Cuando la ROM del CPC inicializa la ROM paralela del disco en ROM, a tiempo de
encendido, reserva 256 octetos (Hex 100) de memoria para su uso exclusivo como un
área de trabajo. Este área de memoria no será alterada por la ROM. Cuando se in-
troduce en la ROM un comando aplicable al disco en ROM, las rutinas del CPC llaman
a la rutina apropiada de manejo del disco en ROM, y le pasa la dirección del área
de trabajo en el registro IY. Por esta razón se especifica el uso del área de tra-
bajo como desplazamientos relativos al contenido del registro IY.

    USO DEL AREA DE TRABAJO RELATIVO AL CONTENIDO DEL REGISTRO IY A LA ENTRADA

   Todos los apuntadores y contadores se almacenan con el octeto de orden infe-
rior en primer lugar.

IY+#30-#32 CAS IN OPEN area para salvar entrada bloque salto (lo usa RD.OPEN)
IY+#33-#35 CAS IN CHAR area para salvar entrada bloque salto (lo usa RD.OPEN)
IY+#36-#38 contiene la última dirección de OPENIN - la pone RD.OPEN
IY+#39-#3C contiene la última dirección de INCHAR - la pone RD.OPEN


IY+#53       = Usada como octeto de señalizadores por RDIR para mostrar que ha
               recibido algunos parámetros de comando
IY+#54 & #55 = Usada por OPENIN para salvar el apuntador a la memoria intermedia
IY+#60       = Memoria intermedia en la que la subrutina LENTRY coloca la entrada
               del hasta directorio que ha sido requerida
IY+#7F
                  El uso de este área es el siguiente
IY+#60          = Octeto marcador = "!" si es una entrada válida (#FF si no lo es)

IY+#61 a IY+#70 = Nombre del fichero en 16 octetos
IY+#71 a IY+#72 = Dirección de carga elegida (octeto inferior en primer lugar)
        IY+#73 = Código de tipo de fichero:

                                              00=Versión ASCII de programa en BASIC
                                              01=Binario
                                              02=Vuelco de pantalla
                                              03=Fichero ASCII


         IY+#74 = Longitud de los datos (octeto inferior)
         IY+#75 = Longitud de los datos (octeto superior)
         IY+#76 = Contador de dirección local (octeto inferior) usado por RDLOAD
         IY+#77 = Contador de dirección local (octeto superior) usado por RDLOAD
         IY+#78 = Contador contenido mem. inter. (inferior) usado por RDINCHAR
         IY+#79 = Contador contenido mem. inter. (superior) usado por RDINCHAR
         IY+#7A = Dirección apuntador mem. inter. (inferior) usado por RDINCHAR
         IY+#7B = Dirección apuntador mem. inter. (superior) usado por RDINCHAR

IY+#7C-#7F Sin asignar
```

Fig 3.82 Uso del área de trabajo del disco en ROM

Lo primero que hace RDLOAD es asegurarse de que el comando incluye un número de ROM y una dirección de memoria donde cargar el contenido de la ROM, si no lo tiene, lanza un mensaje de error a la pantalla. Lo siguiente que hace es comprobar si el número que le ha pasado es legal. Esto es, si no es cero (ya que este está reservado para la ROM

Direcciones de los registros

Registros PPI	Uso	Dirección de E/S
Puerta A	Dirección local baja	FAE0
Puerta B	Dirección local alta	FAE1
Puerta C	Registro de selección de ROM	FAE2
Control	Control del PPI	FAE3
--	Registro de lectura de datos	FAE8

La puerta C forma los 8 bits superiores de una dirección de 24 bits. Esto significa que se pueden direccionar hasta cuatro megaoctetos de ROM en un esquema de disco en ROM expandido totalmente.

Estructura de datos del disco en ROM

Estas notas especifican el formato de una entrada en la ROM de directorio del disco en ROM, y el uso del área de trabajo que se reserva para el uso de los módulos del disco en ROM.

Formato de la entrada de directorio

Por cada una de las 255 ROMs posibles en el esquema del disco en ROM, hay una entrada de 32 octetos en la ROM 0, que es la ROM de directorio. A medida que se añaden ROMs al disco en ROM, el directorio se debe actualizar. Cada entrada tiene el siguiente formato:

```
Octeto 0     = Octeto marcador. Se pone con "!" (valor Hex 21)
Octeto 1-16  = Nombre del contenido - relleno con espacios
Octeto 17-18 = Dirección de carga elegida. (octeto inferior en primer lugar)
Octeto 19    = Tipo de datos
                              0=Un programa BASIC en ASCII
                              1=Binario
                              2=Vuelco de pantalla
                              3=Fichero ASCII que no es un programa
                            4-FF=Sin definir
Octeto 20-21   = Longitud de datos en ROM (octeto inferior en primer lugar)
Octeto 22-31 = Indefinidos - reservado para uso furturo.
```

Direccionando la ROM de directorio para obtener una entrada

Cuando se accede a la ROM directorio, el registro de selección de la ROM debe contener, obviamente, cero. Después se deben poner los dos registros de dirección local de acuerdo con las siguientes reglas:

Bits de dirección 13, 14 y 15	deben ponerse todos a cero
Bits de dirección 5 a 12	deben ponerse con el número de la ROM cuya entrada se desea cargar. Será un número entre 1 y #FF
Bits de dirección 0 a 4	Deben ponerse con la dirección dentro de la entrada, para seleccionar uno de los treinta octetos que componen cada entrada.

Fig 3.83 Hoja informativa del disco en ROM

que contiene el directorio), y que no es mayor de 255 - el máximo número de ROM. Si el número de ROM es legal, la siguiente comprobación consiste en ver si la entrada dentro del directorio, que describe la ROM que se va a cargar, es válida. Si no es válida, lanza un mensaje de error a la pantalla. La comprobación final es ver si el contenido de la ROM cabe dentro de la memoria en la dirección especificada. Esto implica añadir a la dirección deseada la longitud que aparece en la entrada del directorio, si el resultado excede Hex FFFF (la memoria máxima del CPC), aparece en la pantalla el mensaje de error "** Cargar en dirección especificada rebasa la memoria! **". Una vez que se han pasado todas estas comprobaciones se carga el contenido de la ROM dentro de la memoria, en la dirección especificada por el usuario.

El punto principal a tener en cuenta cuando se usa RDLOAD, es que el cargador binario no tiene en cuenta si la memoria está ya asignada por el CPC, el disco en ROM o el BASIC. Por lo tanto hacemos énfasis al usuario para que se asegure que el HIMEM del BASIC se ha modificado, para que haya suficiente memoria disponible como para que quepan todos los datos que se van a cargar. Si carga datos sobre las variables del sistema, la máquina fallará sin remisión. Uno de los usos más efectivos de RDLOAD es la carga de gráficos directamente en la memoria de pantalla, es sumamente rápido y con varias pantallas salvadas en el disco en ROM se abre la posibilidad de crear ciertos juegos rápidos y sofisticados. Por lo tanto, RDLOAD no es difícil de usar, pero tenga cuidado.

```
1800  RDLOAD:   CP   #2          ; Entrada A es contador parámetros, ver si es 2
1810            JR   Z,LOAD01    ; Si, saltar
1820            LD   IX,ERR1     ; No, error
1830            CALL OUTPUT      ; Imprimir el error
1840            JP   LDEXIT      ; E irse
1850  ERR1:     DEFB #0A,#0A,#0D
1860            DEFM "** Necesita número de ROM y dirección de carga **"
1870            DEFB #0A,#0D
1880            DEFM "** Por ejemplo |RD.LOAD,3,&4E20 carga la ROM 3 ** "
1890            DEFB #0A,#0A,#0D,#FF
1900  LOAD01:   PUSH HL
1910            PUSH DE
1920            PUSH IX          ; Hacer sitio
1930            LD   A,(IX+3)    ; Obtener octeto superior del número de ROM
1940            CP   #0          ; Ver si es cero
1950            JR   NZ,MERR2    ; Saltar si no lo es
1960            LD   A,(IX+2)    ; Obtener octeto inferior del número de ROM
1970            CP   #0          ; Ver si es cero
1980            JR   NZ,LOAD02   ; Saltar si no lo es
1990  MERR2:    LD   IX,ERR2     ; Ocurrió un error
2000  COMMOP:   CALL OUTPUT      ; Hacer la salida
2010            POP  IX
2020            POP  DE          ; Restaurar la pila
2030            POP  HL
2040            JP   LDEXIT      ; E irse
2050  ERR2:     DEFB #0A,#0A,#0D
2060            DEFM "** Número de ROM debe ser de 1 a 255 ** "
2070            DEFB #0A,#0A,#0D,#FF
2080  LOAD02:   LD   D,(IX+2)    ; Obtener octeto inferior del número de ROM
2090            CALL LENTRY      ; Cargar la entrada del directorio
```

```
2100            LD   A,(IY+#60)  ; Obtener el octeto indicador
2110            CP   "!"         ; Ver si es válida
2120            JR   Z,LOAD03    ; Si, saltar
2130            LD   IX,ERR3     ; No, mensaje de error número tres
2140            JR   COMMOP      ; Usar la salida común
2150  ERR3:     DEFB #0A,#0A,#0D
2160            DEFM "** Este zócalo ROM está vacío **   "
2170            DEFB #0A,#0A,#0D,#FF
2180  LOAD03:   LD   H,(IX+1)    ; Cargar la dirección en HL
2190            LD   L,(IX)
2200            PUSH DE          ; D contiene el número de ROM - salvarlo
2210            LD   E,(IX+#74)  ; Obtener longitud de la entrada del directorio
2220            LD   D,(IX+#75)
2230            ADD  HL,DE       ; Ver si cabe el contenido de la ROM
2240            POP  DE          ; Recargar DE
2250            JR   NC,LOAD04   ; Si no sobrepasa, tofo va bien
2260            LD   IX,ERR4     ; Si lo hace, mandar mensaje de error
2270            JP   COMMOP      ; Usar la salida común
2280  ERR4:     DEFB #0A,#0A,#0D
2290            DEFM "** Cargar en dirección especificada rebasa la memoria! **"
2300            DEFB #0A,#0A,#0D,#FF
2310  LOAD04:   LD   (IY+#74),00
2320            LD   (IY+#75),00 ; Poner contador de dirección local a cero
2330            LD   BC,ROMSEL       ; Preparar el registro de selección de ROM
2340            OUT  (C),D       ; Con el contenido de D (número de ROM)
2350            LD   H,(IX+1)    ; Poner en HL la dirección en la que se va
2360            LD   L,(IX)      ; a cargar
2370  LOAD05:   LD   BC,ADDRLO
2380            LD   A,(IY+#76)
2390            OUT  (C),A
2400            LD   BC,ADDRHI   ; Mandar la dirección local
2410            LD   A,(IY+#77)  ; Obtener el octeto superior
2420            OUT  (C),A       ; Mandar el octeto superior
2430            LD   BC,DATAP
2440            IN   A,(C)       ; Tomar un octeto de datos de la ROM
2450            LD   (HL),A      ; Almacenarlo
2460            INC  HL          ; Incrementar el apuntador
2470            INC  (IY+#76)    ; Incrementar la dirección local
2480            JR   NZ,LOAD06   ; Si no es cero, saltar
2490            INC  (IY+#77)    ; Incrementar octeto superior dirección local
2500  LOAD06:   LD   A,(IY+#76)  ; Obtiene octeto inferior dirección local
2510            CP   (IY+#74)    ; Ver si dir local es igual a longitud
2520            JR   NZ,LOAD05   ; No, otro bucle
2530            LD   A,(IY+#77)  ; Obtiene octeto superior dirección local
2540            CP   (IY+#75)    ; Ver si coincide
2550            JR   NZ,LOAD05   ; No, repetir de nuevo
2560  LOAD07:   LD   IX,DUNNIT   ; Al llegar aquí, hemos terminado
2570            JP   COMMOP      ; Mandar mensaje DUNNIT y salir
2580  DUNNIT:   DEFB #0A,#0D,#0A
2590            DEFM "** Cargada ROM requerida **  "
2600            DEFB #0A,#0D,#FF
2610  !DEXIT:   RET              ; Todo está hecho, retornar
2620            LD   A,(IY+#77)  ; Si no, obtener dirección local alta
2630            CP   (IY+#75)    ; Comparar con longitud alta
2640            JR   NX,LOAD05   ; Bucle si no coincide
```

Fig 3.84 Listado del programa RDLOAD

Cuando diseñé el disco en ROM, pensé que la mejor forma de implementar el catálogo de su contenido sería tener un área pequeña de cabecera en cada EPROM individual. Posteriores reflexiones sobre esta idea revelaron que no era la forma más eficiente de hacerlo. Como ya hemos dicho anteriormente, uno de los usos más efectivos del disco en ROM es la rapidez de carga de pantallas gráficas. Como la pantalla del CPC ocupa 16K, se requerían 16K octetos de ROM para almacenar la información de una pantalla completa. Si tuviéramos que robar 32 octetos de este espacio para la cabecera de la ROM, el efecto sería que tendríamos una pantalla incompleta, ya sea con varios puntos de menos o con puntos vacíos situados aleatoriamente en la pantalla. Esto me llevó a la alternativa de usar una de las ROM de la tarjeta para contener el directorio de las demás.

El formato de una entrada del directorio, y cómo se direcciona, se ha detallado en la figura 3.83. El único inconveniente de tener una ROM dedicada al directorio es que, siempre que se añade una nueva ROM, la ROM directorio se debe actualizar para que refleje los detalles de la nueva ROM. Esto no es un gran problema, ya que el tipo de datos que se mantienen en el disco en ROM será permanente, y por lo tanto, la actualización del directorio será, después de la fase de preparación inicial, poco frecuente. Ya hemos visto la forma de usar el programador de EPROM para actualizar el contenido de la ROM directorio. Vuelva a la figura 3.80, que muestra el listado del programa BASIC que, cuando se monta la ROM directorio sobre el zócalo ZIF del programador de EPROM, añade una entrada para un número específico de ROM y después lo introduce en la ROM en la dirección indicada.

La ROM directorio debe ser una 2764, ya que solamente necesitamos 8192 octetos (256 veces 32 = 8192). Como en todo el disco en ROM no son necesarias ROM de alta velocidad, así que se pueden usar versiones más baratas sin problemas.

La razón para explicar todo lo relativo a la distribución del directorio del disco en ROM en este punto del texto, es que la siguiente rutina del 'software' del disco en ROM es RDDIR, que produce un listado del directorio basándose en el contenido de la ROM directorio. La figura 3.85 es el listado de la rutina RDDIR. Hay dos formas disponibles del comando desde el BASIC. Estas son "RD.DIR" que produce un listado del directorio sobre la pantalla del CPC, y "RD.DIR,P" que lista el directorio en la pantalla y sobre la impresora. Usted puede teclear cualquier cosa después de "RD.DIR" para imprimir el directorio, el programa no analiza los caracteres extra.

```
560  RDDIR:    PUSH AF          ; Hacer sitio
570            PUSH DE
580            PUSH BC
590            PUSH HL
600            PUSH IX
610            PUSH AF          ; Salvar AF un momento
620            LD   A,2         ; Poner MODE 2
630            CALL #BC0E       ; Llamar al la rutin SCR SET MODE
640            POP  AF
650            LD   E,A         ; Salvar cuenta de parámetros en E
660  RDDIR5:   LD   BC,ROMSEL   ; BC apunta la registro de selección de ROM
670            LD   A,#0
680            OUT  (C),A       ; Asegurarse que se selecciona la ROM cero
690            LD   BC,ADDRHI   ; BC apunta al registro ADDRHI
```

```
700             LD   IX,HEADER   ; IX apunta al principio del texto de cabecera
710             CP   E           ; Ver si B era cero a la entrada
720             CALL NZ,PRINTR   ; No, sacar a impresora
730             LD   IX,HEADER   ; Reapuntar IX a la pantalla
740             CALL OUTPUT      ; Sacar a pantalla usando OUTPUT
750             LD   D,1         ; Usar D como contador de carga
760 RDDIR1:     CALL LENTRY      ; Obtener una entrada de la ROM directorio
770 FORM:       LD   A,"!"       ; Ver si es válida la entrada
780             CP   (IY+#60)    ; Comparar "!" con e primer octeto
790             JP   NZ,RDDIR4   ; Si no es igual, saltar
800             PUSH IY
810             POP  IX          ; Copiar IY sobre IX por la pila
820             LD   BC,#80
830             ADD  IX,BC       ; IX apunta a la memoria intermedia de salida
840             LD   A,D         ; Obtener el número de la entrada del directorio
850             CALL HEXAS       ; Convertir en ASCII
860             LD   (IX),H      ; Introducir el octeto alto
870             LD   (IX+1),L    ; y el bajo dentro de la memoria intermedia
880             LD   BC,2
890             ADD  IX,BC       ; Actualizar apuntador mem. int. de salida
900             LD   B,6         ; Seis espacios
910             CALL SPACE
920 ; Ahora IX está en IY+#88 - Y la memoria intermedia en la que se
930 ; cargó comienza en IY+#60, por lo tanto IX+#27 apunta al segundo
940 ; octeto de la entrada cargada,que es el nombre del fichero.
950             LD   B,16        ; Copiar 16 caracteres
960 NAMET:      LD   A,(IX+#27)  ; Hacer trasferencia octetos de nombre
970             LD   (IX),A      ; Almacenarlo en la memoria intermedia
980             INC  IX          ; Actualizar el apuntador
990             DJNZ NAMET       ; Repetir
1000            LD   B,7         ; Ahora 7 espacios
1010            CALL SPACE
1020            LD   A,(IY+#75)  ; Obtener longitud (octeto alto)
1030            CALL HEXAS       ; Convertir a ASCII
1040            LD   (IX),H      ; Almacenar el octeto superior
1050            LD   (IX+1),L    ; y el inferior
1060            LD   BC,#2
1070            ADD  IX,BC       ; Actualizar el apuntador
1080            LD   A,(IY+#74)  ; Obtener longitud (octeto superior)
1090            CALL HEXAS       ; Convertir también
1100            LD   (IX),H
1110            LD   (IX+1),L
1120            LD   BC,#2
1130            ADD  IX,BC
1140            LD   B,7         ; Ahora 7 espacios en memoria intermedia
1150            CALL SPACE
1160 ; Ahora tenemos el tipo de fichero como texto
1170            LD   A,(IY+#73)  ; Obtener octeto de tipo de fichero
1180            CP   #0          ; Ver si es BASIC
1190            JR   NZ,BINCHK   ; No, saltar
1200            LD   HL,BAS      ; Si, HL apunta al texto
1210            CALL XFER        ; Trasferirlo
1220            JR   RDDIR8      ; Salir
1230 BAS:       DEFM "BASIC"
1240            DEFB #80
1250 BINCHK:    CP   #1          ; Ver si es binario
1260            JR   NZ,SDCHK    ; No, saltar
1270            LD   HL,BIN      ; Si, HL apunta al texto
1280            CALL XFER        ; Trasferirlo
```

```
1290            JR   RDDIR8      ; Salir
1300  BIN:      DEFM "BIN   "
1310            DEFB #80
1320  SDCHK:    CP   #2          ; Ver si es vuelco de pantalla
1330            JR   NZ,ASCCHK   ; No, saltar
1340            LD   HL,SD       ; Si, HL apunta al texto
1350            CALL XFER        ; Trasferirlo
1360            JR   RDDIR8      ; Salir
1370  SD:       DEFM "Vuelco Pant"
1380            DEFB #80
1390  ASCCHK:   CP   #3          ; Ver si es ASCII
1400            JR   NZ,WHAT     ; No, saltar
1410            LD   HL,ASC      ; Si, HL apunta al texto
1420            CALL XFER        ; Trasferirlo
1430            JR   RDDIR8      ; Salir
1440  ASC:      DEFM "ASCII"
1450            DEFB #80
1460  WHAT:     LD   HL,UNKNOW   ; Cargar HL con el texto
1470            CALL XFER        ; Trasferirlo
1480            JR   RDDIR8      ; Salir
1490 UNKNOW     DEFM "Descon."
1500            DEFB #80
1510  RDDIR8:   LD   (IX),#0D
1520            LD   (IX+1),#0A
1530            LD   (iX+2),#A0
1540            PUSH IY          ; Copiar IY en IX de nuevo
1550            POP  IX
1560            LD   BC,#80
1570            ADD  IX,BC       ; IX apunta a la mem. int. de salida
1580            LD   A,E         ; Obtener señalizador de parámetros
1590            CP   #0          ; Ver si es cero
1600            CALL NZ,PRINTR   ; Si no es cero, sacar a impresora
1610            PUSH IY
1620            POP  IX
1630            LD   BC,#80      ; Hacer lo mismo de nuevo
1640            ADD  IX,BC       ; Salida a la pantalla
1650            CALL OUTPUT      ; Sacar a la pantalla
1660  RDDIR4:   LD   A,1         ; Incrementar D (Contador de entradas)
1670            ADD  A,D         ; Sumar D
1680            LD   D,A         ; Poner el resultado en D
1690            JP   NZ,RDDIR1   ; repetir de nuevo si no ha terminado
1700  RDDIR7:   POP  IX          ; Si pasa de 255, recuperar los registros
1710            POP  HL
1720            POP  BC
1730            POP  DE
1740            POP  AF
1750            RET              ; Y volver al BASIC
1760  HEADER: DEFM   "ROM     Nombre              Longitud"
1770          DEFM "     Tipo."
1780          DEFB #0D,#0A
1790          DEFB #FF
```

Fig 3.85 Listado del programa RDDIR

Lo primero que hace RDDIR es llamar a la rutina de la ROM del CPC para poner MODE 2. Después selecciona la ROM cero, y decide si tiene que mandar el listado solamente a la pantalla o también a la impresora. A continuación imprime una cabecera para cada parámetro (Nombre

de fichero, Longitud, etc.). Después carga sucesivamente cada una de las 255 posibles entradas, imprimiendo los detalles en la pantalla y en la impresora, según se haya seleccionado. Las entradas se cargan en orden descendente. Si después de cargar una entrada, el primer octeto es igual a "!", significa que la entrada es válida, y se imprime su contenido. Gran parte de la rutina RDDIR se encarga de averiguar el tipo de fichero y sacarlo en forma de texto. Refiérase a la figura 3.83 para ver el significado de cada valor posible del octeto de tipo de fichero.

RDDIR usa varias rutinas de la librería de subrutinas del disco en ROM. La principal de ellas es HEXAS, que convierte un octeto simple en dos caracteres ASCII, y LENTRY, que carga en memoria la entrada del directorio cuyo número recibe en el registro D. RDDIR usa también PRINTR y OUTPUT, que manda el carácter que recibe en el registro A, a la impresora y la pantalla, respectivamente.

Los comandos finales realizan la carga y ejecución de los programas del disco en ROM, y se usan desde el BASIC. Sin embargo, puede ser de utilidad explicar cómo se carga una versión ASCII de un programa BASIC, antes de que nos metamos a describir las rutinas RDOPEN, OPENIN, RDCLOS e INCHAR. Mi agradecimiento a Cliff Lawson de Amsoft por su asistencia en este área.

Cuando usted le dice al BASIC que cargue un programa que ha sido almacenado en disco o en cinta en formato ASCII, él llama a la rutina CAS IN OPEN, suministrándole la dirección del nombre del fichero que desea cargar, junto con la dirección de un área de memoria de 2048 octetos donde desea que se cargue el primer bloque del fichero. CAS IN OPEN lo hace y devuelve al BASIC la dirección de la tabla de cabecera que contiene toda la información del fichero. En este punto, la versión 1.0 del BASIC, comprueba solamente el tipo de fichero que le ha pasado. Los valores inválidos en otros campos no causarán mensajes de error, y pueden causar resultados extraños en los siguientes pasos. A continuación, el BASIC usa otra rutina llamada CAS IN CHAR para pasar el contenido del área de 2048, octeto a octeto, al BASIC. CAS IN CHAR pasa los 2048 octetos y, si el fichero consta de más de un bloque, carga el siguiente bloque desde la cinta (o desde el disco), y repite el proceso. Cuando se han cargado todos los bloques y el número de octetos, indicados en la entrada de la longitud en la cabecera del fichero, se han pasado al BASIC, CAS IN CHAR manda un carácter de fin de fichero (Hex 1A) y ciertos señalizadores al BASIC, que hace terminar el proceso y llama a la rutina CAS IN CLOSE.

Todo el tiempo durante el que se están mandando caracteres por medio de CAS IN CHAR, el BASIC ejecuta un proceso de conversión a la versión abreviada interna.

Esta no es una descripción precisa de cómo carga el BASIC los ficheros, pero está basada en la información encontrada en el manual de la ROM de Amsoft, SOFT 158, y en la observación y experimentación.

```
3750  RDOPEN:   PUSH AF
3760            PUSH HL
3770            LD   A,(#BC77)   ; Obtener y salvar las entradas del bloque
3780            LD   (IY+#30),A  ; de saltos de CAS IN OPEN
3790            LD   A,(#BC78)   ; Salvarlas en IY+#30-32
3800            LD   (IY+#31),A
```

```
3810            LD   A,(#BC79)
3820            LD   (IY+#32),A
3830            LD   A,(#BC80)   ; Hacer lo mismo para las entradas
3840            LD   (IY+#33),A  ; de CAS IN CHAR
3850            LD   A,(#BC81)   ; Salvarlas en IY+#33-35
3860            LD   (IY+#34),A
3870            LD   A,(#BC82)
3880            LD   (IY+#35),A
3890            LD   A,(#BC7A)   ; Salvar el contenido originale de
3900            LD   (IY+#3C),A  ; las posiciones de retorno
3910            LD   A,(#BC83)
3920            LD   (IY+#3D),A
3930            LD   A,#C9       ; Y poner una instrucción RET
3940            LD   (#BC7A),A   ; en ellas
3950            LD   (#BC83),A
3960            LD   A,#DF
3970            LD   (#BC77),A   ; Poner las instrucciones de rearranque
3980            LD   (#BC80),A   ; en el lugar de los bloques de salto
3990            LD   HL,OPENIN   ; Ahora debemos hacer direcciones lejanas
4000            LD   (IY+#36),L  ; en el área de trabajo.
4010            LD   (IY+#37),H
4020            LD   H,ROMNUN    ; Colocar la dirección y el número
4030            LD   (IY+#38),H  ; de la ROM
4040            LD   (IY+#3B),H
4050            LD   HL,INCHAR   ; Y tambien RD.INCHAR
4060            LD   (IY+#39),L
4070            LD   (IY+#3A),H
4080            PUSH IY
4090            POP  HL          ; Copiar IY en HL
4100            PUSH DE          ; Ahora necesitamos DE, salvarlo
4110            LD   DE,#36      ; Usar DE para hacer que HL apunte a
4120            ADD  HL,DE       ; la primera dirección lejana
4130            LD   (#BC78),HL  ; Ponerla en el bloque de salto
4140            LD   DE,#3       ; HL apunta a la segunda
4150            ADD  HL,DE       ; dirección lejana
4160            LD   (#BC81),HL  ; Ponerla en el bloque de salto
4170            POP  DE          ; Ya hemos hecho todo
4180            POP  HL          ; así que restauramos
4190            POP  AF          ; y se hace
4200            RET              ; el retorno
```

Fig 3.86 Listado del programa RDOPEN

El comando RD.OPEN del disco en ROM, hace que se modifiquen las entradas de los bloques de salto de CAS IN OPEN y CAS IN CHAR (o sus equivalentes de disco). La figura 3.86 muestra la rutina RDOPEN. Las entradas se hacen sobre las rutinas OPENIN e INCHAR del disco en ROM. Debido a la forma en que se accede a las rutinas del disco en ROM, también se debe alterar un octeto de cada entrada de los bloques de salto de CAS IN CLOSE y CAS IN DIRECT, para hacer que las llamadas a estas rutinas, una vez modificadas, hagan un retorno sin realizar ninguna función. Los valores anteriores se almacenan en el área de trabajo del disco en ROM, para que puedan ser restaurados cuando se lance un comando RD.CLOSE. RDCLOS aparece listado en la figura 3.87. Tanto OPENIN como INCHAR utilizan una subrutian de la librería del disco en ROM llamada TWOKAY. Como indica su nombre, esta rutina carga 2K octetos desde la ROM seleccionada. A la entrada espera que el registro IY apunte al área de trabajo del disco en ROM. El registro

HL debe apuntar a la memoria intermedia, donde TWOKAY debe cargar los 2K octetos. Toma los valores que encuentra en IY+#76 a IY+#77 (el octeto inferior en primer lugar) como dirección local desde la que debe empezar la carga. Después de la ejecución de TWOKAY, la dirección local se habrá incrementado en 2048, así como el contenido del registro HI.(TWOKAY aparece en el listado de la librería de subrutinas del disco en ROM, al final de esta sección).

```
4210  RDCLOS:   PUSH AF          ; Salvar AF
4220            LD   A,(IY+#30)  ; Tomar un octeto
4230            LD   (#BC77),A   ; restaurarlo
4240            LD   A,(IY+#31)  ; Tomar un octeto
4250            LD   (#BC78),A   ; restaurarlo
4260            LD   A,(IY+#32)  ; y así sucesivamente ....
4270            LD   (#BC79),A
4280            LD   A,(IY+#33)
4290            LD   (#BC80),A
4300            LD   A,(IY+#34)
4310            LD   (#BC81),A
4320            LD   A,(IY+#35)
4330            LD   (#BC82),A   ; Hasta que se hayan restaurado todos
4340            LD   A,(IY+#3C)
4350            LD   (#BC7A),A
4360            LD   A,(IY+#3D)
4370            LD   (#BC83),A
4380            POP  AF          ; Recargar AF y retornar
4390            RET
```

Fig 3.87 Listado del programa RDCLOS

Después de ejecutar RDOPEN, cualquier intento por parte del BASIC para cargar desde cinta o disco, dará como resultado el uso de OPENIN e INCHAR, para cargar el fichero (s existe) desde el disco en ROM, de forma similar a la que lo hacen las rutinas de la ROM del CPC desde disco o cinta. La figura 3.88 es el listado de OPENIN, que carga los primeros 2048 octetos, INCHAR (listada en la figura 3.89) introduce los caracteres de uno en uno en el BASIC, rellenando la memoria intermedia cuando se requiera, y repitiendo el ciclo hasta que encuentra un octeto con el bit siete a uno, que señala al BASIC que se ha llegado al fin de fichero y que la carga está completa.

```
4400  OPENIN:   LD   A,B           ; Mirar si es válido el
4410            CP   #0            ; nombre del fichero
4420            JR   NZ,OP001      ; Si, saltar
4430            LD   IX,ERR01      ; No, cargar el mensaje de error
4440            JP   BADEND        ; y saltar a la rutina de error
4450  ERR01:    DEFB #0A,#0D,#0A
4460            DEFM '** DEBE especificar un nombre fichero en ROMdisk ** '
4470            DEFB #0A,#0D,#FF
4480  OP001:    LD   (IY+#54),E    ; Salvar dirección de mem. int.
4490            LD   (IY+#55),D
4500            LD   D,1           ; Empezr con entrada 1 del directorio
4510  OP002:    CALL LENTRY        ; Cargar la entrada
4520            LD   A,(IY+#60)    ; Obtener el octeto de marca
4530            CP   "!"           ; Ver si es una entrada válida
4540            JR   Z,OP005       ; Si, saltar
```

```
4550 OP004:    INC  D           ; Incrementar el contador de entradas
4560           JR   NZ,OP002    ; Si no es el final, repetir
4570           LD   IX,ERR02    ; Fichero no encontrado
4580 BADEND:   CALL OUTPUT      ; Rutinae común de error
4590           LD   A,#0        ; Cargar cero en A
4600           ADD  A,#0        ; Limpiar acarreo
4610           CP   #0          ; Poner señalizador de cero
4620           RET              ; volver
4630 ERR02:    DEFB #0A,#0A,#0D
4640           DEFM "** Fich. no encontrado **"
4650           DEFB #0A,#0D,#FF
4660 OP005:    PUSH IY
4670           PUSH DE          ; Salvar número de entrada
4680           LD   DE,#0061    ; IY apunta al nombre del fichero
4690           ADD  IY,DE
4700           POP  DE          ; Recargar DE
4710 ; ** Ahora IY apunta al nombre del fichero cargado, y HL todavía
4720 ; ** apunta al nombre de fichero requerido. B=long nombre del fich.
4730           CALL COMPAR      ; Llamar a subrutina de comparar
4740           POP  IY          ; Recargar IY
4750           JR   NZ,OP004    ; Saltar si falla la comparación
4760           LD   A,(IY+#73)  ; Obtener tipo de fichero
4770           CP   #0          ; Ver si es ASCII (tipo=0)
4780           JR   Z,OP003     ; Si es ASCII, saltar
4790           LD   IX,ERR03    ; No, error
4800           JR   BADEND      ; y salir
4810 ERR03:    DEFB #0A,#0D
4820           DEFM "** Error tipo - este fichero no es ASCII BASIC **  "
4830           DEFB #0A,#0D,#FF
4840 OP003:    LD   BC,ROMSEL
4850           OUT  (C),D       ; Seleccionar la ROM deseada
4860           LD   E,(IY+#54)  ; Recargar en DE la dirección de la mem. int.
4870           LD   D,(IY+#55)
4880           LD   (IY+#71),E  ; ponerlo en la dirección elegida
4890           LD   (IY+#72),D
4900           LD   (IY+#7A),E  ; Apunt. mem. int. =base  mem. int.
4910           LD   (IY+#7B),D
4920           PUSH DE          ; Copia en HL la dirección de la mem. int.
4930           POP  HL
4940           LD   (IY+#76),0  ; Cero el contador de dirección LOCAL
4950           LD   (IY+#77),0
4960           CALL TWOKAY      ; Llamar al cargador de 2K
4970           PUSH IY          ; Copiar IY en HL
4980           POP  HL
4990           LD   DE,#61
5000           ADD  HL,DE
5010           LD   DE,#170     ; Posición =Hex 170 y HL apunta a cabecera
5020           LD   C,(IY+#74)  ; BC longitud fichero
5030           LD   B,(IY+#75)
5040           LD   A,#16       ; A = tipo de fichero
5050           CP   #0          ; Limpiar señalizador Z
5060           SCF              ; Poner acarreo
5070           RET              ; y retornar
```

Fig 3.88 Listado del programa OPENIN

Con el disco en ROM, puede usar los comandos LOAD, MERGE, CHAIN y OPENIN, pero tenga en cuenta lo que dijimos anteriormente, que cuando

use el disco en ROM como canal 9 (#9) le dará resultados inesperados. LOAD, MERGE y CHAIN funcionarán con programas BASIC salvados en ASCII en el disco en ROM.

En la figura 3.90 aparecen los listados de las subrutinas que se usan en varios módulos del disco en ROM. Estas están ampliamente explicadas en los comentarios de los listados, y en el texto.

Finalmente tenemos la figura 3.91 con el listado completo del ensamblaje del 'software' del disco en ROM.

```
5080  INCHAR:   PUSH HL          ; Salvar HL
5090            LD   L,(IY+#7A)  ; HL apunta a la mem. int.
5100            LD   H,(IY+#7B)
5110            LD   A,(HL)      ; Toma un carácter de la mem. int.
5120            INC  HL          ; Incrementa el apuntador
5130            LD   (IY+#7A),L  ; Salva el apuntador
5140            LD   (IY+#7B),H
5150            DEC  (IY+#78)    ; Decrementa la cantidad en la mem. int.
5160            JR   NZ,INCH01   ; Si no es cer, salta
5170            DEC  (IY+#79)    ; Si es cero, decrementa el contador alto
5180            JR   NZ,INCH01   ; Salta si todavía no es cero
5190            LD   L,(IY+#71)  ; Si es cero, HL=base mem. int.
5200            LD   H,(IY+#72)  ; Y después debemos
5210            PUSH AF          ; llamar a TWOKAY para
5220            CALL TWOKAY      ; rellenar la mem. int. de nuevo
5230            LD   A,(IY+#71)  ; Apuntador apunta a la dirección
5240            LD   (IY+#7A),A  ; base de la mem. int.
5250            LD   A,(IY+#72)
5260            LD   (IY+#7B),A
5270            POP  AF          ; Recuperar AF
5280  INCH01:   BIT  7,A         ; Ver si está puesto el séptimo bit
5290            JR   Z,INCH02    ; No, saltar
5300            LD   A,#1A       ; Cargado el fichero: mandar el EOF
5310            CP   #10         ; Limpiar el señalizador Z
5320            JR   INCH03      ; e ir a la salida común
5330  INCH02:   SCF              ; Poner acarreo
5340            CP   #FF         ; Limpiar señalizador Z
5350  INCH03:   POP  HL          ; Salida común - restaurar HL
5360            RET              ; y retornar
```

Fig 3.89 Listado del programa INCHAR

```
2645  ; **      RDLIB - Librería de subrutinas del disco en ROM       **
2650  LENTRY:   PUSH BC          ; Necesitamos BC, HL y DE, limpiarlos
2660            PUSH HL
2670            PUSH DE
2680            LD   BC,ROMSEL   ; Seleccionar ROM cero
2690            LD   A,0
2700            OUT  (C),A       ; Hecho.
2710            LD   A,D         ; Esta rutina carga una entrada de directorio
2720            RRC  A           ; D es el número de la entrada
2730            RRC  A           ; Ver texto para detalles de como se direcciona
2740            RRC  A           ; una entrada. Crear octeto inferior dirección
2750            AND  #E0         ; Eliminar bits no requeridos
2760            LD   BC,ADDRLO
2770            OUT  (C),A       ; Mandar el octeto inferior de la dirección
```

```
2780            LD   A,D          ; Recargar el número de ROM original
2790            AND  #F8          ; Enmascarar los tres bits inferiores
2800            SRL  A            ; Desplazar tres bits
2810            SRL  A
2820            SRL  A            ; Mandar la dirección alta a la puerta.
2830            LD   BC,ADDRHI    ; BC apunta a ella
2840            OUT  (C),A        ; Y hacer el OUT
2850            PUSH IY           ; Mover el contenido de IY a HL
2860            POP  HL
2870            LD   A,L
2880            ADD  A,#60
2890            LD   L,A          ; Poner el resultado de nuevo
2900            JR   NC,LENT00    ; Si no hay acarreo, saltar
2910            INC  H
2920 LENT00:    LD   A,32         ; cargar 32 octetos- A=bucle
2930            LD   BC,DATAP     ; BC apunta a la puerta de datos
2940 LENT01:    IN   E,(C)        ; Obtener un octeto
2950            LD   (HL),E       ; Almacenarlo
2960            DEC  A            ; Ver si han terminado los 32
2970            JR   Z,LENT02     ; Si, saltar
2980            INC  HL           ; Incrementar apuntador a mem. int.
2990            PUSH HL           ; Salvar apuntador a mem. int.
3000            LD   BC,ADDRLO    ; BC apunta a registro dirección baja
3010            IN   L,(C)        ; Obtener la dirección baja
3020            INC  L            ; Incrementarla
3030            OUT  (C),L        ; voverla a mandar
3040            JR   NZ,LENT03    ; Si no es cero, saltar
3050            LD   BD,ADDRHI    ; BC apunta a octeto superior de dirección
3060            IN   L,(C)        ; Traerlo
3070            INC  L            ; Incrementarlo
3080            OUT  (C),L        ; Devolverlo
3090 LENT03:    LD   BC,DATAP     ; BC apunta al registro de datos del sig. bucle
3100            POP  HL           ; Restaurar apuntador a mem. int.
3110            JR   LENT01       ; Repetir de nuevo
3120 LENT02:    POP  DE
3130            POP  HL
3140            POP  BC           ; Recargar
3150            RET               ; y salir
3160 OUTPUT:    LD   A,(IX)       ; Obtener un octeto
3170            BIT  7,A          ; Ver si es el marcador EOB
3180            RET  NZ           ; Si, retornar
3190            CALL #BB5A        ; No, mandar el carácter
3200            INC  IX           ; Incrementar el apuntador de la mem. int.
3210            JR   OUTPUT       ; y repetir de nuevo
3220 PRINTR:    CALL #BD2E        ; Ver si la impresora está ocupada
3230            JR   NC,PRINT1    ; No, saltar
3240            CALL #BB1B        ; Si, comprobar el teclado
3250            RET  C            ; Cortar si se ha pulsado alguna tecla
3260            JR   PRINTR       ; Esperar
3270 PRINT1:    LD   A,(IX)       ; Obtener un carácter
3280            BIT  7,A          ; Ver si es EOB
3290            RET  NZ           ; Si, retornar
3300            CALL #BD31        ; Mandar el carácter a la impresora
3310            INC  IX           ; Incrementar el apuntador
3320            JR   PRINTR       ; y repetir de nuevo
3330 HEXAS:     LD   H,A          ; Salvar en H el octeto original
3340            AND  #0F          ; dejar solo la mitad inferior
3350            OR   #30          ; Convertir a ASCII
3360            CP   #3A          ; Ver si es alfabético
```

```
3370            JP   M,HIGH      ; No, saltar sobre el ajuste
3380            ADD  A,#7        ; Covertir en Hex
3390  HIGH:     LD   L,A         ; Poner el dígito terminado en L
3400            LD   A,H         ; Tomar el octeto original
3410            SRL  A           ; Poner la mitad superior en la inferior
3420            SRL  A
3430            SRL  A
3440            SRL  A
3450            OR   #30
3460            CP   #3A         ; Ver si es alfabético
3470            JP   M,DONE      ; No, hecho
3480            ADD  A,#7        ; Ajustarlo como antes
3490  DONE:     LD   H,A         ; Salvar la cosa terminada en H
3500            RET              ; y retornar
3510  SPACE:    LD   (IX),#20    ; Poner un espacio
3520            INC  IX          ; Incrementar el apuntador
3530            DJNZ SPACE       ; Repetir hasta que esté hecho
3540            RET              ; y retornar
3550  XFER:     LD   A,(HL)
3560            BIT  7,A         ; Ver si es EOT
3570            RET  NZ          ; Si, retornar
3580            LD   (IX),A      ; Colocar el octeto en la mem. int. de salida
3590            INC  IX          ; Incrementar apuntador salida
3600            INC  HL          ; Incrementar apuntador mem. int. de texto
3610            JR   XFER        ; Repetir de nuevo
3620  COMPAR:   PUSH HL          ; Salvar registros
3630            PUSH BC
3640            PUSH IY
3650  CP001:    LD   A,(IY)      ; Obtener un carácter
3660            CP   (HL)        ; Comaprarlo
3670            JR   NZ,CP002    ; si no coincide, salir
3680            INC  HL          ; Incrementar los apuntadores
3690            INC  IY
3700            DJNZ CP001       ; Repetir si hay más
3710  CP002:    POP  IY          ; Terminado, recargar los registros y salir
3720            POP  BC          ;
3730            POP  HL
3740            RET
```

Fig 3.90 Listado del programa RDLIB

```
                1 ; ** Ensamblaje del software del disco en ROM
C000            100       ORG  #C000
                110 *T+ ROMDISK.OBJ
                120 ;
                130 ;  ** Software del disco en ROM
                140 ;
FAE0            150 ADDRLO EQU  #FAE0
FAE1            160 ADDRHI EQU  #FAE1
FAE2            170 ROMSEL EQU  #FAE2
FAE3            180 CTRLP  EQU  #FAE3
FAE8            190 DATAP  EQU  #FAE8
                200 ;
0006            210 ROMNUM EQU  6
C000  01        220        DEFB #1    ; ROM PARALELA
C001  010100    230        DEFB #1,#1,#0   ; Marca 1 Vers 1 Rev 0
C004  15C0      240        DEFW NAMTAB          ; apuntador tab.nom.
C006  C339C0    250        JP   RDINIT
```

```
C009  C30EC4     260         JP   RDOPEN
C00C  C380C4     270         JP   RDCLOS
C00F  C375C0     280         JP   RDDIR
C012  C386C1     290         JP   RDLOAD
C015  52442049   300 NAMTAB DEFM "RD INI"   ; Ent. init.
C01B  D4         310        DEFB "T"+#80
C01C  52442E4F   320        DEFM "RD.OPE"
C022  CE         330        DEFB "N"+#80
C023  52442E43   340        DEFM "RD.CLOS"
C02A  C5         350        DEFB "E"+#80
C02B  52442E44   360        DEFM "RD.DI"
C030  D2         370        DEFB "R"+80
C031  52442E4C   380        DEFM "RD.LOA"
C037  C4         390        DEFB "D"+80
C038  00         400        DEFB 0          ; fin tab. nom.
C039  25         410 RDINIT DEC  H
C03A  01E3FA     420        LD   BC,CTRLP
C03D  3E80       430        LD   A,#80
C03F  ED79       440        OUT  (C),A
C041  0B         450        DEC  BC
C042  3E00       460        LD   A,#0
C044  ED79       470        OUT  (C),A
C046  DD214FC0   480        LD   IX,INIMES
C04A  CD9FC3     490        CALL OUTPUT
C04D  1824       500        JR   INIDUN
C04F  0A0D       510 INIMER DEFB #0A,#0D
C051  2A2A2052   520        DEFM "** ROMdisk v1.2 iniciado **   "
C06F  0A0D0DFF   530        DEFB #0A,#0D,#0A,#FF
C073  37         540 INIDUN SCF
C074  C9         550        RET
C075  F5         560 RDDIR  PUSH AF
C076  D5         570        PUSH DE
C077  C5         580        PUSH BC
C078  E5         590        PUSH HL
C079  DDE5       600        PUSH IX
C07B  F5         610        PUSH AF
C07C  3E02       620        LD   A,
C07E  CD0EBC     630        CALL #BC0E
C081  F1         640        POP  AF
C082  5F         650        LD   E,A
C083  01E2FA     660 RDDIR5 LD   BC,ROMSEL
C086  3E00       670        LD   A,#0
C088  ED79       680        OUT  (C),A
C08A  01E1FA     690        LD   BC,ADDRHI
C08D  DD2193C1   700        LD   IX,HEADER
C091  BB         710        CP   E
C092  C4ACC3     720        CALL NZ,PRINTR
C095  DD2193C1   730        LD   IX,HEADER
C099  CD9FC3     740        CALL OUTPUT
C09C  1601       750        LD   D,1
C09E  CD47C3     760 RDDIR1 CALL LENTRY
C0A1  3E21       770 FORM   LD   A,"!"
C0A3  FDBE60     780        CP   (IY+#60)
C0A6  C285C1     790        JP   NZ,RDDIR4
C0A9  FDE5       800        PUSH IY
C0AB  DDE1       810        POP  IX
C0AD  018000     820        LD   BC,#80
C0B0  DD09       830        ADD  IX,BC
C0B2  7A         840        LD   A,D
```

```
C0B3  CDC4C3     850        CALL HEXAS
C0B6  DD7400     860        LD   (IX),H
C0B9  DD7501     870        LD   (IX+1),L
C0BC  010200     880        LD   BC,2
C0BF  DD09       890        ADD  IX,BC
C0C1  0606       900        LD   B,6
C0C3  CDE5C3     910        CALL SPACE
C0C6  0610       950        LD   B,16
C0C8  DD7ED9     960 NAMET  LD   A,(IX+#27)
C0CB  DD7700     970        LD   (IX),A
C0CE  DD23       980        INC  IX
C0D0  10F6       990        DJNZ NAMET
C0D2  0607      1000        LD   B,7
C0D4  CDE5C3    1010        CALL SPACE
C0D7  FD7E75    1020        LD   A,(IY+#75)
C0DA  CDC4C3    1030        CALL HEXAS
C0DD  DD7400    1040        LD             (IX),H
C0E0  DD7501    1050        LD   (IX+1),L
C0E3  010200    1060        LD   BC,#2
C0E6  DD09      1070        ADD  IX,BC
C0E8  FD7E74    1080        LD   A,(IY+#74)
C0EB  CDC4C3    1090        CALL HEXAS
C0EE  DD7400    1100        LD   (IX),H
C0F1  DD7501    1110        LD   (IX+1),L
C0F4  010200    1120        LD   BC,#2
C0F7  DD09      1130        ADD  IX,BC
C0F9  CDE5C3    1150        CALL SPACE
                1160 ; Tipo de fichero como texto
C0FC  FD7E73    1170        LD   A,(IY+#73)
C0FF  FE00      1180        CP   #0
C101  200E      1190        JR   NZ,BINCHK
C103  210BC1    1200        LD   HL,BAS
C106  CDEEC3    1210        CALL XFER
C109  1853      1220        JR   RDDIR8
C10B  42415349  1230 BAS    DEFM "BASIC"
C110  80        1240        DEFB #80
C111  FE01      1250 BINCHK CP   #1
C113  200F      1260        JR   NZ,SDCHK
C115  211DC1    1270        LD   HL,BIN
C118  CDEEC3    1280        CALL XFER
C11B  1841      1290        JR   RDDIR8
C11D  42694E20  1300 BIN    DEFM "BIN
C123  80        1310        DEFB #80
C124  FE02      1320 SDCHK  CP   #2
C126  2014      1330        JR   NZ,ASCCHK
C128  2130C1    1340        LD   HL,SD
C12B  CDEEC3    1350        CALL XFER
C12E  182E      1360        JR   RDDIR8
C130  50414E54  1370 SD     DEFM "PANTALLA
C13B  80        1380        DEFB #80
C13C  FE03      1390 ASCCHK CP   #3
C13E  200E      1400        JR   NZ,WHAT
C140  2148C1    1410        LD   HL,ASC
C143  CDEEC3    1420        CALL XFER
C146  1816      1430        JR   RDDIR8
C148  41534349  1440 ASC    DEFM "ASCII"
C14D  80        1450        DEFB #80
C14E  2156C1    1460 WHAT   LD   HL,UNKNOW
C151  CDEEC3    1470        CALL XFER
```

```
C154  1808      1480        JR   RDDIR8
C156  44657363  1490 UNKNOW DEFM "Descon."
C15D  80        1500        DEFB #80
C15E  DD36000D  1510 RDDIR8 LD   (IX),#0D
C162  DD36010A  1520        LD   (IX+1),#0A
C166  DD3602A0  1530        LD   (iX+2),#A0
C16A  FDE5      1540        PUSH IY
C16C  DDE1      1550        POP  IX
C16E  018000    1560        LD   BC,#80
C171  DD09      1570        ADD  IX,BC
C173  7B        1580        LD   A,E
C174  FE00      1590        CP   #0
C176  C4ACC3    1600        CALL NZ,PRINTR
C179  FDE5      1610        PUSH IY
C17B  DDE1      1620        POP  IX
C17D  018000    1630        LD   BC,#80
C180  DD09      1640        ADD  IX,BC
C182  CD9FC3    1650        CALL OUTPUT
C185  3E01      1660 RDDIR4 LD   A,1
C187  82        1670        ADD  A,D
C188  57        1680        LD   D,A
C189  C29EC0    1690        JP   NZ,RDDIR1
C18C  DDE1      1700 RDDIR7 POP  IX
C18E  E1        1710        POP  HL
C18F  C1        1720        POP  BC
C190  D1        1730        POP  DE
C191  F1        1740        POP  AF
C192  C9        1750        RET
C193  524F4D20  1760 HEADER DEFM "ROM  Nombre     Long. "
C1A9  20202020  1770        DEFM "     Tipo."
C1B3  0D0A      1780        DEFB #0D,#0A
C1B5  FF        1790        DEFB #FF
C1B6  FE02      1800 RDLOAD CP   #2
C1B8  2857      1810        JR   Z,LOAD01
C1BA  DD21C4C1  1820        LD   IX,ERR1
C1BE  CD9FC3    1830        CALL OUTPUT
C1C1  C33EC3    1840        JP   LDEXIT
C1C4  0A0A0D    1850 ERR1   DEFB #0A,#0A,#0D
C1C7  2A2A204E  1860        DEFM "** Nec. no. ROM + dir. carg. **"
C1E5  0A0D      1870        DEFB #0A,#0D
C1E7  2A2A2045  1880        DEFM "** EJ. |RD.LOAD,3,&4E20 carga ROM 3 ** "
C20D  0A0A0DFF  1890        DEFB #0A,#0A,#0D,#FF
C211  E5        1900 LOAD01 PUSH HL
C212  D5        1910        PUSH DE
C213  DDE5      1920        PUSH IX
C215  DD7E03    1930        LD   A,(IX+3)
C218  FE00      1940        CP   #0
C21A  2007      1950        JR   NZ,MERR2
C21C  DD7E02    1960        LD   A,(IX+2)
C21F  FE00      1970        CP   #0
C221  2033      1980        JR   NZ,LOAD02
C223  DD2131C2  1990 MERR2  LD   IX,ERR2
C227  CD9FC3    2000 COMMOP CALL OUTPUT
C22A  DDE1      2010        POP  IX
C22C  D1        2020        POP  DE
C22D  E1        2030        POP  HL
C22E  C33EC3    2040        JP   LDEXIT
C231  0A0A0D    2050 ERR2   DEFB #0A,#0A,#0D
C234  2A2A204E  2060        DEFM "** Num. ROM entre 1 y 255 **   "
```

```
C252  0A0A0DFF  2070         DEFB #0A,#0A,#0D,#FF
C256  DD5602    2080 LOAD02  LD   D,(IX+2)
C259  CD47C3    2090         CALL LENTRY
C25C  FD7E60    2100         LD   A,(IY+#60)
C25F  FE21      2110         CP   "!"
C261  282C      2120         JR   Z,LOAD03
C263  DD2169C2  2130         LD   IX,ERR3
C267  18BE      2140         JR   COMMOP
C269  0A0A0D    2150 ERR3    DEFB #0A,#0A,#0D
C26C  2A2A2045  2160         DEFM "** Esta ROM no está grabada **     "
C288  0A0A0DFF  2170         DEFB #0A,#0A,#0D,#FF
C28F  DD6601    2180 LOAD03  LD   H,(IX+1)
C292  DD6E00    2190         LD   L,(IX)
C295  D5        2200         PUSH DE
C296  FD5E74    2210         LD   E,(IX+#74)
C299  FD5675    2220         LD   D,(IX+#75)
C29C  19        2230         ADD  HL,DE
C29D  D1        2240         POP  DE
C29E  3032      2250         JR   NC,LOAD04
C2A0  DD21A7C2  2260         LD   IX,ERR4
C2A4  C327C2    2270         JP   COMMOP
C2A7  0A0A0D    2280 ERR4    DEFB #0A,#0A,#0D
C2AA  2A2A204C  2290         DEFM "** La dirección rebasa la memoria! **"
C2CE  0A0A0DFF  2300         DEFB #0A,#0A,#0D,#FF
C2D2  FD367600  2310 LOAD04  LD   (IY+#74),00
C2D6  FD367700  2320         LD   (IY+#75),00
C2DA  01E2FA    2330         LD   BC,ROMSEL
C2DD  ED51      2340         OUT  (C),D
C2DF  DD6601    2350         LD   H,(IX+1)
C2E2  DD6E00    2360         LD   L,(IX)
C2E5  01E0FA    2370 LOAD05  LD   BC,ADDRLO
C2E8  FD7E76    2380         LD   A,(IY,#76)
C2EB  ED79      2390         OUT  (C),A
C2ED  01E1FA    2400         LD   BC,ADDRHI
C2F0  FD7E77    2410         LD   A,(IY+#77)
C2F3  ED79      2420         OUT  (C),A
C2F5  01E8FA    2430         LD   BC,DATAP
C2F8  ED78      2440         IN   A,(C)
C2FA  77        2450         LD   (HL),A
C2FB  23        2460         INC  HL
C2FC  FD3476    2470         INC  (IY+#76)
C2FF  2003      2480         JR   NZ,LOAD06
C301  FD3477    2490         INC  (IY+#77)
C304  FD7E76    2500 LOAD06  LD   A,(IY+#76)
C307  FDBE74    2510         CP   (IY+#74)
C30A  20D9      2520         JR   NZ,LOAD05
C30C  FD7E77    2530         LD   A,(IY+#77)
C30F  FDBE75    2540         CP   (IY+#75)
C312  20D1      2550         JR   NZ,LOAD05
C314  DD211BC3  2560 LOAD07  LD   IX,DUNNIT
C318  C327C2    2570         JP   COMMOP
C31B  0A0D0A    2580 DUNNIT  DEFB #0A,#0D,#0A
C31E  2A2A2043  2590         DEFM "** Cargada ROM requerida **  "
C33B  0A0DFF    2600         DEFB #0A,#0D,#FF
C33E  C9        2610 LDEXIT  RET
C33F  FD7E77    2620         LD   A,(IY+#77)
C342  FDBE75    2630         CP   (IY+#75)
C345  209E      2640         JR   NX,LOAD05
C347  C5        2650 LENTRY  PUSH BC
```

```
C348  E5        2660         PUSH HL
C349  D5        2670         PUSH DE
C34A  01E0FA    2680         LD   BC,ROMSEL
C34D  3E00      2690         LD   A,0
C34F  C9        2700         OUT  (C),A
C355  7A        2710         LD   A,D
C351  CB0F      2720         RRC  A
C353  CB0F      2730         RRC  A
C355  CB0F      2740         RRC  A
C357  E6E0      2750         AND  #E0
C359  01E0FA    2760         LD   BC,ADDRLO
C35C  ED79      2770         OUT  (C),A
C35E  7A        2780         LD   A,D
C35F  E6F8      2790         AND  #F8
C361  CB3F      2800         SRL  A
C363  CB3F      2810         SRL  A
C365  CB3F      2820         SRL  A
C367  01E1FA    2830         LD   BC,ADDRHI
C36A  ED79      2840         OUT  (C),A
C36C  FDE5      2850         PUSH IY
C36E  E1        2860         POP  HL
C36F  7D        2870         LD   A,L
C370  C660      2880         ADD  A,#60
C372  6F        2890         LD   L,A
C373  3001      2900         JR   NC,LENT00
C375  24        2910         INC  H
C376  3E20      2920 LENT00  LD   A,32
C378  01E8FA    2930         LD   BC,DATAP
C37B  ED58      2940 LENT01  IN   E,(C)
C37D  73        2950         LD   (HL),E
C37E  3D        2960         DEC  A
C37F  281A      2970         JR   Z,LENT02
C381  23        2980         INC  HL
C382  E5        2990         PUSH HL
C383  01E0FA    3000         LD   BC,ADDRLO
C386  ED68      3010         IN   L,(C)
C388  2C        3020         INC  L
C389  ED69      3030         OUT  (C),L
C38B  2008      3040         JR   NZ,LENT03
C38D  01E1FA    3050         LD   BD,ADDRHI
C390  ED68      3060         IN   L,(C)
C392  2C        3070         INC  L
C393  ED69      3080         OUT  (C),L
C395  01E8FA    3090 LENT03  LD   BC,DATAP
C398  E1        3100         POP  HL
C399  18E0      3110         JR   LENT01
C39B  D1        3120 LENT02  POP  DE
C39C  E1        3130         POP  HL
C39D  C1        3140         POP  BC
C39E  C9        3150         RET
C39F  DD7E00    3160 OUTPUT  LD   A,(IX)
C3A2  CB7F      3170         BIT  7,A
C3A4  C0        3180         RET  NZ
C3A5  CD5ABB    3190         CALL #BB5A
C3A8  DD23      3200         INC  IX
C3AA  18F3      3210         JR   OUTPUT
C3AC  CD2EBD    3220 PRINTR  CALL #BD2E
C3AF  3006      3230         JR   NC,PRINT1
C3B1  CD1BBB    3240         CALL #BB1B
```

```
C3B4  D8        3250        RET  C
C3B5  18F5      3260        JR   PRINTR
C3B7  DD7E00    3270 PRINT1 LD   A,(IX)
C3BA  CB7F      3280        BIT  7,A
C3BC  C0        3290        RET  NZ
C3BD  CD31BD    3300        CALL #BD31
C3C0  DD23      3310        INC  IX
C3C2  18E8      3320        JR   PRINTR
C3C4  67        3330 HEXAS  LD   H,A
C3C5  E60F      3340        AND  #0F
C3C7  F630      3350        OR   #30
C3C9  FE3A      3360        CP   #3A
C3CB  FAD0C3    3370        JP   M,HIGH
C3CE  C607      3380        ADD  A,#7
C3D0  6F        3390 HIGH   LD   L,A
C3D1  7C        3400        LD   A,H
C3D2  CB3F      3410        SRL  A
C3D4  CB3F      3420        SRL  A
C3D6  CB3F      3430        SRL  A
C3D8  CB3F      3440        SRL  A
C3DA  F630      3450        OR   #30
C3DC  FE3A      3460        CP   #3A
C3DE  FAE3C3    3470        JP   M,DONE
C3E1  C607      3480        ADD  A,#7
C3E3  67        3490 DONE   LD   H,A
C3E4  C9        3500        RET
C3E5  DD360920  3510 SPACE  LD   (IX),#20
C3E9  DD23      3520        INC  IX
C3EB  10F8      3530        DJNZ SPACE
C3ED  C9        3540        RET
C3EE  7E        3550 XFER   LD   A,(HL)
C3EF  CB7F      3560        BIT  7,A
C3F1  C0        3570        RET  NZ
C3F2  DD7700    3580        LD   (IX),A
C3F5  DD23      3590        INC  IX
C3F7  23        3600        INC  HL
C3F8  18F4      3610        JR   XFER
C3FA  E5        3620 COMPAR PUSH HL
C3FB  C5        3630        PUSH BC
C3FC  FDE5      3640        PUSH IY
C3FE  FD7E00    3650 CP001  LD   A,(IY)
C401  BE        3660        CP   (HL)
C402  2005      3670        JR   NZ,CP002
C404  23        3680        INC  HL
C405  FD23      3690        INC  IY
C407  10F5      3700        DJNZ CP001
C409  FDE1      3710 CP002  POP  IY
C40B  C1        3720        POP  BC
C40C  E1        3730        POP  HL
C40D  C9        3740        RET
C40E  F5        3750 RDOPEN PUSH AF
C40F  E5        3760        PUSH HL
C410  3A77BC    3770        LD   A,(#BC77)
C413  FD7730    3780        LD   (IY+#30),A
C416  3A78BC    3790        LD   A,(#BC78)
C419  FD7731    3800        LD   (IY+#31),A
C41C  3A79BC    3810        LD   A,(#BC79)
C41F  FD7732    3820        LD   (IY+#32),A
C422  3A80BC    3830        LD   A,(#BC80)
```

```
C425  FD7733    3840         LD   (IY+#33),A
C428  3A81BC    3850         LD   A,(#BC81)
C42B  FD7734    3860         LD   (IY+#34),A
C42E  3A82BC    3870         LD   A,(#BC82)
C431  FD7735    3880         LD   (IY+#35),A
C434  3A7ABC    3890         LD   A,(#BC7A)
C437  FD773C    3900         LD   (IY+#3C),A
C43A  3A83BC    3910         LD   A,(#BC83)
C43D  FD773D    3920         LD   (IY+#3D),A
C440  3EC9      3930         LD   A,#C9
C442  327ABC    3940         LD   (#BC7A),A
C445  3283BC    3950         LD   (#BC83),A
C448  3EDF      3960         LD   A,#DF
C44A  3277BC    3970         LD   (#BC77),A
C44D  3280BC    3980         LD   (#BC80),A
C450  21B3C4    3990         LD   HL,OPENIN
C453  FD7536    4000         LD   (IY+#36),L
C456  FD7437    4010         LD   (IY+#37),H
C459  2606      4020         LD   H,ROMNUN
C45B  FD7438    4030         LD   (IY+#38),H
C45E  FD743B    4040         LD   (IY+#3B),H
C461  219BC5    4050         LD   HL,INCHAR
C464  FD7539    4060         LD   (IY+#39),L
C467  FD743A    4070         LD   (IY+#3A),H
C46A  FDE5      4080         PUSH IY
C46C  E1        4090         POP  HL
C46D  D5        4100         PUSH DE
C46E  113600    4110         LD   DE,#36
C471  19        4120         ADD  HL,DE
C472  2278BC    4130         LD   (#BC78),HL
C475  110300    4140         LD   DE,#3
C478  19        4150         ADD  HL,DE
C479  2281BC    4160         LD   (#BC81),HL
C47C  D1        4170         POP  DE
C47D  E1        4180         POP  HL
C47E  F1        4190         POP  AF
C47F  C9        4200         RET
C480  F5        4210 RDCLOS  PUSH AF
C481  FD7E30    4220         LD   A,(IY+#30)
C484  3277BC    4230         LD   (#BC77),A
C487  FD7E31    4240         LD   A,(IY+#31)
C48A  3278BC    4250         LD   (#BC78),A
C48D  FD7E32    4260         LD   A,(IY+#32)
C490  3279BC    4270         LD   (#BC79),A
C493  FD7E33    4280         LD   A,(IY+#33)
C496  3280BC    4290         LD   (#BC80),A
C499  FD7E34    4300         LD   A,(IY+#34)
C49C  3281BC    4310         LD   (#BC81),A
C49F  FD7E35    4320         LD   A,(IY+#35)
C4A2  3282BC    4330         LD   (#BC82),A
C4A5  FD7E3C    4340         LD   A,(IY+#3C)
C4A8  327ABC    4350         LD   (#BC7A),A
C4AB  FD7E3D    4360         LD   A,(IY+#3D)
C4AE  3283BC    4370         LD   (#BC83),A
C4B1  F1        4380         POP  AF
C4B2  C9        4390         RET
C4B3  78        4400 OPENIN  LD   A,B
C4B4  FE00      4410         CP   #0
C4B6  2028      4420         JR   NZ,OP001
```

```
C4B8  DD21BFC4  4430         LD   IX,ERR01
C4BC  C3F9C4    4440         JP   BADEND
C4BF  0A0D0A    4450 ERR01   DEFB #0A,#0D,#0A
C4C2  2A2A2046  4460         DEFM '** FALTA nombre fichero ** '
C4DD  0A0DFF    4470         DEFB #0A,#0D,#FF
C4E0  FD7354    4480 OP001   LD   (IY+#54),E
C4E3  FD7255    4490         LD   (IY+#55),D
C4E6  1601      4500         LD   D,1
C4E8  CD47C3    4510 OP002   CALL LENTRY
C4EB  FD7E60    4520         LD   A,(IY+#60)
C4EE  FE21      4530         CP   "!"
C4F0  2830      4540         JR   Z,OP005
C4F2  14        4550 OP004   INC  D
C4F3  20F3      4560         JR   NZ,OP002
C4F5  DD2103C5  4570         LD   IX,ERR02
C4F9  CD9FC3    4580 BADEND  CALL OUTPUT
C4FC  3E00      4590         LD   A,#0
C4FE  C600      4600         ADD  A,#0
C500  FE00      4610         CP   #0
C502  C9        4620         RET
C503  0A0A0D    4630 ERR02   DEFB #0A,#0A,#0D
C506  2A2A2046  4640         DEFM "** Fich. no encontrado **"
C51F  0A0DFF    4650         DEFB #0A,#0D,#FF
C522  FDE5      4660 OP005   PUSH IY
C524  D5        4670         PUSH DE
C525  116100    4680         LD   DE,#0061
C528  FD19      4690         ADD  IY,DE
C52A  D1        4700         POP  DE
C52B  CDFAC3    4730         CALL COMPAR
C52E  FDE1      4740         POP  IY
C530  20C0      4750         JR   NZ,OP004
C532  FD7E73    4760         LD   A,(IY+#73)
C535  FE00      4770         CP   #0
C537  2828      4780         JR   Z,OP003
C539  DD213FC5  4790         LD   IX,ERR03
C53D  18BA      4800         JR   BADEND
C53F  0A0D      4810 ERR03   DEFB #0A,#0D
C541  2A2A2046  4820         DEFM "** Fich. no es ASCII BASIC **"
C55E  0A0DFF    4830         DEFB #0A,#0D,#FF
C561  01E2FA    4840 OP003   LD   BC,ROMSEL
C564  ED51      4850         OUT  (C),D
C566  FD5E54    4860         LD   E,(IY+#54)
C569  FD5655    4870         LD   D,(IY+#55)
C56C  FD7371    4880         LD   (IY+#71),E
C56F  FD7272    4890         LD   (IY+#72),D
C572  FD737A    4900         LD   (IY+#7A),E
C575  FD727B    4910         LD   (IY+#7B),D
C578  D5        4920         PUSH DE
C579  E1        4930         POP  HL
C57A  FD367600  4940         LD   (IY+#76),0
C57E  FD367700  4950         LD   (IY+#77),0
C582  CDDAC5    4960         CALL TWOKAY
C585  FDE5      4970         PUSH IY
C587  E1        4980         POP  HL
C588  116100    4990         LD   DE,#61
C58B  19        5000         ADD  HL,DE
C58C  117001    5010         LD   DE,#170
C58F  FD4E74    5020         LD   C,(IY+#74)
C592  FD4675    5030         LD   B,(IY+#75)
```

```
C595  3E16      5040        LD   A,#16
C597  FE00      5050        CP   #0
C599  37        5060        SCF
C59A  C9        5070        RET
C59B  E5        5080 INCHAR PUSH HL
C59C  FD6E7A    5090        LD   L,(IY+#7A)
C59F  FD667B    5100        LD   H,(IY+#7B)
C5A2  7E        5110        LD   A,(HL)
C5A3  23        5120        INC  HL
C5A4  FD757A    5130        LD   (IY+#7A),L
C5A7  FD747B    5140        LD   (IY+#7B),H
C5AA  FD3578    5150        DEC  (IY+#78)
C5AD  201C      5160        JR   NZ,INCH01
C5AF  FD3579    5170        DEC  (IY+#79)
C5B2  2017      5180        JR   NZ,INCH01
C5B4  FD6E71    5190        LD   L,(IY+#71)
C5B7  FD6672    5200        LD   H,(IY+#72)
C5BA  F5        5210        PUSH AF
C5BB  CDDAC5    5220        CALL TWOKAY
C5BE  FD7E71    5230        LD   A,(IY+#71)
C5C1  FD777A    5240        LD   (IY+#7A),A
C5C4  FD7E72    5250        LD   A,(IY+#72)
C5C7  FD777B    5260        LD   (IY+#7B),A
C5CA  F1        5270        POP  AF
C5CB  CB7F      5280 INCH01 BIT  7,A
C5CD  2806      5290        JR   Z,INCH02
C5CF  3E1A      5300        LD   A,#1A
C5D1  FE10      5310        CP   #10
C5D3  1803      5320        JR   INCH03
C5D5  37        5330        INCH02     SCF
C5D6  FEFF      5340        CP   #FF
C5D8  E1        5350 INCH03 POP  HL
C5D9  C9        5360        RET
C5DA  C5        5500 TWOKAY PUSH BC
C5DB  D5        5510        PUSH DE
C5DC  110000    5520        LD   DE,#0000
C5DF  01E0FA    5530 TWLOOP LD   BC,ADDRLO
C5E2  FD7E76    5540        LD   A,(IY+#76)
C5E5  ED79      5550        OUT  (C),A
C5E7  01E1FA    5560        LD   BC,ADDRHI
C5EA  FD7E77    5570        LD   A,(IY+#77)
C5ED  ED79      5580        OUT  (C),A
C5EF  01E8FA    5590        LD   BC,DATAP
C5F2  ED78      5600        IN   A,(C)
C5F4  77        5610        LD   (HL),A
C5F5  FD3476    5620        INC  (IY+#76)
C5F8  2003      5630        JR   NZ,TOOK1
C5FA  FD3477    5640        INC  (IY+#77)
C5FD  23        5650 TOOK1  INC  HL
C5FE  13        5660        INC  DE
C5FF  CB5A      5670        BIT  3,D
C601  28DC      5680        JR   Z,TWLOOP
C603  FD367800  5690        LD   (IY+#78),00
C607  FD367908  5700        LD   (IY+#79),#08
C60B  D1        5710        POP  DE
C60C  C1        5720        POP  BC
C60D  C9        5730        RET
```

```
Pass 2 errors: 00

ADDRHI FAE1  ADDRLO FAE0  ASC    C148  ASCCHK C13C
BADEND C4F9  BAS    C10B  BIN    C11D  VINCHK C111
COMMOP C227  COMPAR C3FA  CP001  C3FE  CP002  C409
CTRLP  FAE3  DATAP  FAE8  DONE   C3E3  DUNNIT C31B
ERR01  C48F  ERR02  C503  ERR03  C53F  ERR1   C1C4
ERR2   C231  ERR3   C269  ERR4   C2A7  FORM   C0A1
HEADER C193  HEXAS  C3C4  HIGH   C3D0  INCH01 C5CB
INCH02 C5D5  INCH03 C5D8  INCHAR C59B  INIDUN C073
INIMES C04F  LDEXIT C33E  LENT00 C376  LENT01 C37B
LENT02 C39B  LENT03 C395  LENTRY C347  LOAD01 C211
LOAD02 C256  LOAD03 C28F  LOAD04 C2D2  LOAD05 C2E5
LOAD06 C304  LOAD07 C314  MERR2  C223  NAMET  C0C8
NAMTAB C015  OP001  C4E0  OP002  C4E8  OP003  C561
OP004  C4F2  OP005  C522  OPENIN C4B3  OUTPUT C39F
PRINT1 C3B7  PRINTR C3AC  RDCLOS C480  RDDIR  C075
RDDIR1 C09E  RDDIR4 C185  RDDIR5 C083  RDDIR7 C18C
RDDIR8 C15E  RDINIT C039  RDLOAD C1B6  RDOPEN C40E
ROMNUM 0006  ROMSEL FAE2  SD     C130  SDCHK  C124
SPACE  C3E5  TOOK1  C5FD  TWLOOP C5DF  TWOKAY C5DA
UNKNOW C156  WHAT   C14E  XFER   C3EE

Table used:   990  form  1109
```

Fig 3.91 Listado del ensamblaje del 'software' del disco en ROM

Conclusión

Con esto concluimos la descripción del 'software' del disco en ROM. Como el disco en ROM usa ciertas facilidades de la ROM del CPC, es bastante complejo, pero le proporciona una facilidad extra para su CPC, que probablemente será de utilidad para muchos de los lectores. Puede cargar pantallas enteras muy rápidamente, los programas BASIC se cargan también muy rápido, y con algún programa extra puede convertir su disco en ROM en una base de datos de información estática para su colección de sellos, discos, libros o cuadros, con un acceso rápido a cuatro megaoctetos potenciales de datos. En la figura 3.92 está la lista de componentes del disco en ROM, y en la figura 3.93 tiene los chips usados y sus patillas de alimentación.

Condensadores

C1-C4	0.1 mfd	10 voltios disco
C5	100 mfd	50 voltios radial

Resistencias

R1-R6	10K	Ohmios 1/4 watio

Semiconductores

E1	7425	2 puertas NOR de 4 entradas
E2	74LS21	2 puertas NAND de 4 entradas
E3	74LS20	2 puertas AND de 4 entradas
E4	74LS05	Inversores Hex con colector abierto
E5	8255A	chip PPI
E6	74LS138	decodificador de 3 a 8 líneas
E7	74LS245	Transceptor octal
TR1	BC107	Transistor

Varios

Zócalos para todos los chips
Conector para el 'bus' de expansión
CONN1 conector para las tarjetas de expansión del disco en ROM
E8-E15: zócalos DIL de 28 patillas
Circuito impreso

Fig 3.92 Lista de componentes del disco en ROM

Número E	Disposit.	Descripción	Patillas aliment.			
			+5V	+12V	-12V	OV
1	7425	2 puertas NOR 4 entradas	14	--	--	7
2	74LS21	2 puertas AND 4 entradas	14	--	--	7
3	74LS20	2 puertas NAND 4 entradas	14	--	--	7
4	74LS05	Inversor Hex C/A	14	--	--	7
5	8255A	Chip PPI	26	--	--	7
6	74LS138	Decodificador de 3 a 8	16	--	--	8
7	74LS245	Transceptor octal	20	--	--	10

Fig 3.93 Patillas de alimentación de los chips de este proyecto

LA TARJETA DE EXPANSIÓN DE ROM

Como Amstrad comprobó al principio de su diseño que 64K de memoria podría ser restrictivo, el diseño de los ordenadores CPC permiten al usuario desactivar la ROM superior (ver capítulo uno) que contiene el BASIC. La forma de hacerlo es mediante el registro de selección de ROM. Este registro es solo de escritura, de ocho bits, está en la dirección DF00 de E/S y está ubicada físicamente en la matriz de puertas del vídeo. El registro se carga con cero, para seleccionar la ROM interna del BASIC. Si se carga con cualquier otro valor, el circuito externo debe usar la señal ROMDIS del 'bus' de expansión para desactivar la ROM interna cuando se lea la memoria en el rango C000 a FFFF. Así, si el registro de selección de ROM contiene cualquier valor que no sea cero, los 16K superiores del mapa de memoria se pueden usar para leer. (Note que la escritura en estas direcciones escribe siempre en la memoria de la pantalla).

Una vez establecido que se puede forzar al BASIC a que libere este espacio de direcciones, el siguiente paso es añadir algo de 'hardware' extra para permitirnos colocar nuestros propios programas en este espacio de direcciones. Además tenemos la ventaja de que el área baja de la ROM (que contiene el sistema operativo del CPC), contiene subrutinas que permiten la intercomunicación entre la ROM del BASIC y otra ROM que podemos conmutar en su lugar. Suena algo complejo, ¿verdad?, En efecto, aunque hay ciertas consideraciones complejas que aplicar, cuando se escriben cierto tipo de programas extra, en ROM (programas que reemplacen el BASIC), es relativamente fácil añadir rutinas que proporcionen comandos extra. Antes de considerar los diferentes tipos de ROM de expansión que se pueden añadir, veamos el último proyecto 'hardware' de este libro, la tarjeta de expansión de ROM.

Detalles del 'hardware'.

La figura 3.94 muestra el diagrama de bloques de la tarjeta de expansión de ROM. Como el registro de selección de ROM no es accesible externamente, tenemos que duplicarlo en la tarjeta. El registro de selección de ROM interno no resulta afectado de ninguna forma. Las señales del 'bus' del CPC se refuerza externamente con unos circuitos llamados 'buffer', ya que vamos a cargarlos bastante y corremos el peligro de que se degrade la calidad de sus señales si no lo hacemos. Las direcciones del 'bus' se conectan dentro de las entradas de dirección de los ocho zócalos de expansión de ROM. Una vez más, para mantener la compatibilidad con el programador de EPROM, y porque caben exactamente en el espacio de direcciones de la ROM del CPC, debemos usar EPROMs 27128 (o 2764 si quiere añadir programas en espacios de 8K), pero a diferencia de las EPROM de la tarjeta del disco en ROM, las que montaremos en la tarjeta de expansión deben ser de la versión más rápida, de 250 nanosegundos.

El decodificador de direcciones está compuesto por dos decodificadores. El primero produce una señal para activar la ROM seleccionada cuando el Z80A lea de las direcciones del rango C000 a FFFF. El segundo produce una señal de sincronismo para el registro de selección de ROM (el nuestro, no el interno) cuando se detecta que se escribe en una dirección de E/S que empieza por Hex DF.

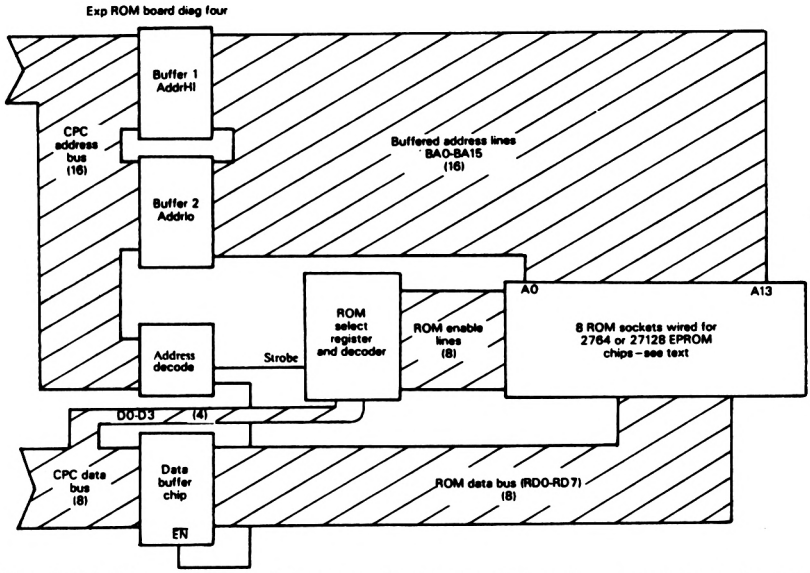


Fig 3.94 Diagrama de bloques de la tarjeta de expansión de ROM

El chip 'buffer' de datos se coloca por las mismas razones que los de direcciones.

El registro de selección de ROM y el decodificador están en el mismo chip, dentro del cual los cuatro bits inferiores del 'bus' de direcciones se enclavan cuando el decodificador de direcciones les da el impulso de sincronismo. El chip usado contiene también un decodificador de cuatro a dieciséis líneas conectado a su circuito interno de enclavamiento.

Descripción del 'hardware' de la tarjeta de expansión de ROM

La figura 3.95 muestra el decodificador de direcciones y registro de selección de ROM. Las puertas E1b y E2b forman el decodificador de direcciones, que detecta cuándo se intenta leer del espacio de direcciones de la ROM superior. Está condicionado por la puerta E7, cuya salida estará en estado lógico alto si se escribe en el registro de selección de ROM algo que no sea cero. La salida de la puerta E2b, cuando es uno lógico se usa para cuatro propósitos. Primero habilita el chip 'buffer' de datos (ver figura 3.96). Segundo pone la señal del 'bus' de expansión DOUT negativa, en estado lógico bajo. Tercero, fuerza la señal del 'bus' de expansión ROMDIS a estado uno lógico, desactivando la ROM interna del CPC, cuando está activa la tarjeta de expansión de ROM.

En la figura 3.95, las puertas E2a y E1a decodifican las cuatro líneas de dirección superiores, y cuando estas contienen Hex D y está en estado bajo la línea IORQ negativa, al mismo tiempo que se pone baja la señal WR negativa, se manda un impulso de sincronismo a E3, que es el chip de enclavamiento del chip decodificador de 4 a 16 lí-

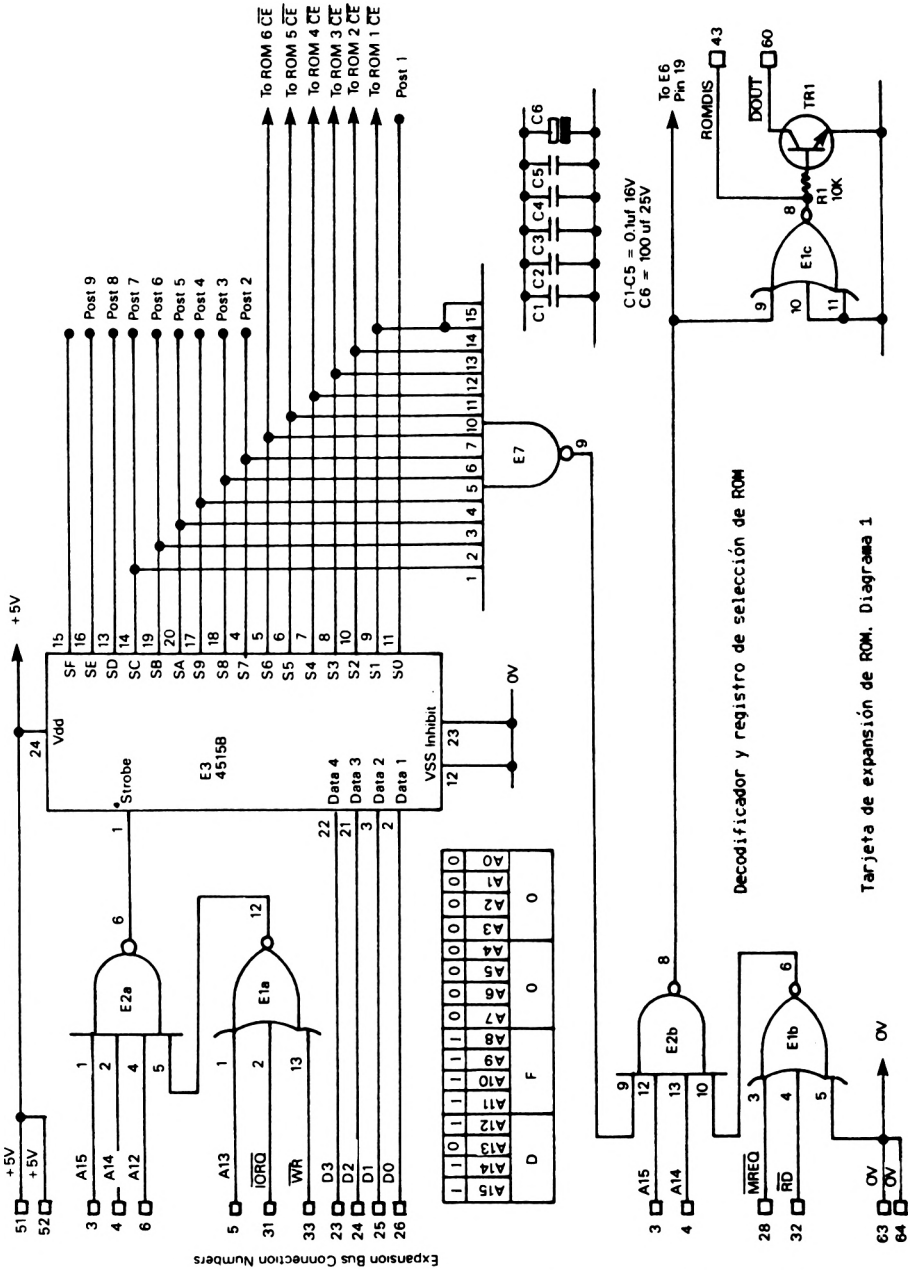


Fig. 3.95: Esquema del decodificador de direcciones de la tarjeta de expansión de ROM.

neas. Está permitido decodificar solamente las cuatro líneas superiores de dirección, porque el registro de selección de ROM es el único dispositivo cuya dirección de E/S comienza por hex D. E3 enclava los cuatro bits inferiores del 'bus' de datos del CPC, cuando su entrada STROBE (sincronismo) se pone baja. Las salidas S correspondientes al dígito retenido en el circuito de enclavamiento de E3 se ponen bajas. Las salidas S están conectadas a la entrada CE negativa de la EPROM, montada en los zócalos de la expansión de ROM. Muchas de las salidas S de E3 terminan en puntos de soldadura, de forma que usted puede escoger la dirección en la que desea emplazar la ROM. Como ya hemos visto, la ROM del BASIC suele ser la ROM cero (aunque si usted usa la ROM cero, las rutinas del CPC siempre activarán su ROM y nunca la del BASIC, por lo tanto, si usa la ROM cero no podrá acceder al BASIC). La posición siete se usa para las rutinas de disco, por lo tanto los zócalos cero y siete tienen sus entradas CE negativas conectadas a punto de soldadura de forma que puede soldar un puente a una de las salidas del decodificador de selección de ROM. Las posiciones de ROM una a seis están conectadas directamente al decodificador.

Los condensadores C1 a C6 desacoplan y filtran la línea de +5 voltios de lo forma usual.

La figura 3.96 muestra los chips de 'buffer' de datos y los de direcciones. Los chips E7 y E5 son 74LS245, que toman las señales desde el 'bus' de expansión del CPC y las pasa a las conexiones de entrada de dirección de los zócalos de ROM. El chip E6 - otro 74LS245 - toma las señales de datos desde los zócalos de las ROM y las pasa al 'bus' del CPC cuando el decodificador de direcciones las activa para hacerlo, como ya se ha descrito. Estos chips 'buffer' representan una carga muy pequeña para el Z80A, que maneja las líneas de direcciones y de datos del CPC, y permiten la posibilidad de cargar las líneas de datos y direcciones del 'bus', sin peligro de afectar a sus niveles lógicos.

El último diagrama es el de la figura 3.97 que muestra cómo se conecta un zócalo típico de ROM. Todas las ROM tiene sus patillas conectadas en paralelo, con la única excepción de la entrada CE negativa, cada una de ellas está conectada a una salida S diferente del decodificador de selección de ROM.

Uso de la expansión de ROM

La conexión de una tarjeta de expansión de ROM en su CPC le abre multitud de posibilidades. Estas irán apareciendo a medida que veamos cada uno de los tipos de ROM que puede reconocer el sistema.

El tipo de ROM más difícil es la ROM preferente. Una ROM preferente contiene un programa que puede asumir el control completo del CPC, haciendo uso de las rutinas del la parte inferior de la ROM del CPC. La ROM del BASIC del CPC es una ROM preferente.

Más simple y muy útil es la ROM secundaria que contiene las rutinas del disco en ROM, que vimos en la descripción del último proyecto. Se pueden conectar ROMs secundarias para obtener facilidades y comandos extra, consiguiendo una extensión del BASIC para añadir comandos con los que controlar 'hardware' exterior. Las ROM secundarias deben ser ubicadas en una dirección de selección entre uno y siete.

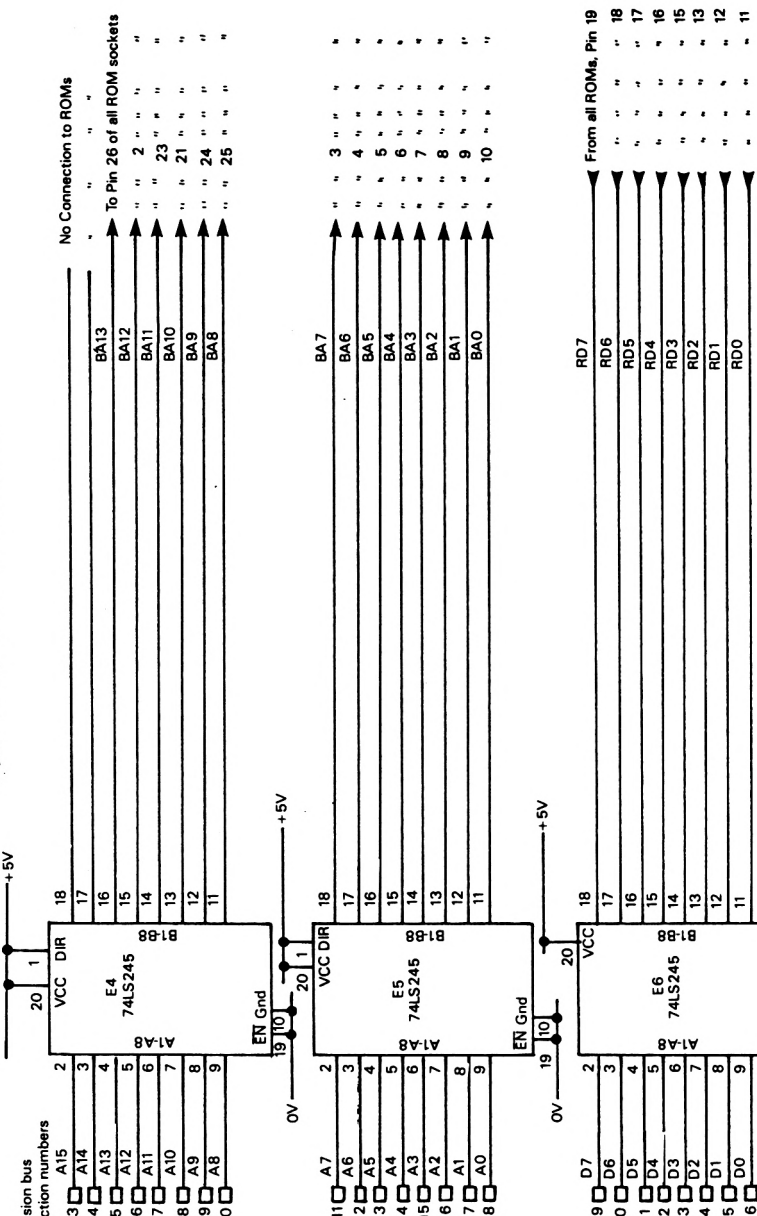


Fig. 3.96: Chips del buffer de datos y direcciones.

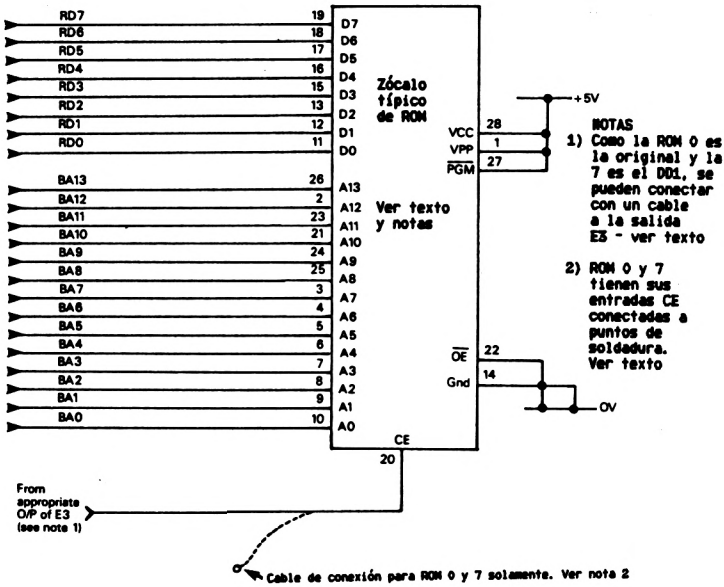


Fig 3.97 Cableado típico de zócalo de ROM en la tarjeta de expansión

El tipo final de ROM que reconoce el CPC es una ROM de expansión. Este tipo contiene programas o datos para cargar en RAM para su ejecución o acceso por otros programas.

Hay un formato estricto para la tabla de cabeceras de cualquier expansión de ROM. Este formato se detalla en SOFT 158 - el manual de la ROM del CPC.

Conclusión

La tarjeta de expansión de ROM le permitirá añadir 'software' permanente a su CPC, y le permitirá también desarrollar sus propios programas para nuevos dispositivos que pueda construir. La figura 3.98 es la lista de componentes necesarios para el proyecto de tarjeta de expansión de la ROM, La figura 3.99 le muestra los chips necesarios junto con sus patillas de alimentación. No debe encontrar problemas para construir el proyecto, el único punto problemático podría ser la selección de la dirección de ROM que va a usar. Si usted tiene más dispositivos extra conectados aparte de los de este libro, comprueba la documentación que tiene de ellos para ver si tienen una ROM dentro, si la tienen evite usar su dirección de selección de ROM para cualquier otra ROM que vaya a usar en la tarjeta de expansión de ROM. Los dos únicos proyectos de este libro que usan una ROM de expansión son el disco en ROM y el proyecto de red de datos -ver capítulo seis.

Condensadores

C1-C5	0.1 mfd 10 voltios disco
C6	100 mfd 50 voltios radial

Resistencias

R1	10K Ohmios 1/4 watio

Semiconductores

E1	74LS27 3 puertas NOR de 3 entradas
E2	74LS20 2 puertas AND de 4 entradas
E3	4515B CMOS decodificador de enclavamiento de 4 a 16 líneas
E4 - E6	74LS245 Transceptor octal
E7	74LS133 1 puerta NAND de 13 entradas
TR1	BC107 Transistor

Varios

Zócalos para todos los chips
Conector para el 'bus' de expansión
Circuito impreso
Puntos de soldadura (10)

Fig. 3.98 Lista de componentes de la tarjeta de expansión de ROM

Número E	Disposit.	Descripción	Patillas aliment. +5V	+12V	-12V	OV
1	74LS27	3 puertas NOR 3 entradas	14	--	--	7
2	74LS20	2 puertas NAND 4 entradas	14	--	--	7
3	4515B	Enclav. de 4 a 6 líneas	24	--	--	12
4	74LS245	Transceptor octal	20	--	--	10
5	74LS245	Transceptor octal	20	--	--	10
6	74LS245	Transceptor octal	20	--	--	10
7	74LS133	1 puerta NAND 13 entradas	14	--	--	7

Fig 3.99 Patillas de conexión de los chips usados en este proyecto

Conclusión del capítulo tres

En este capítulo hemos visto diez proyectos de extensión 'hardware' para añadir a su CPC, mejorar su capacidad, y expandir el rango de usos a los que se puede aplicar. Espero que los construya y disfrute usando por lo menos un par de ellos. Si se encuentra confuso por cualquiera de los términos o conceptos usados en la descripción de los proyectos, consulte primero el capítulo seis, si no encuentra en él la respuesta a sus dudas intente buscarla en alguno de los libros listados en el apéndice dos, y si aún ahí no encuentra la respuesta a sus problemas, busque en una librería especializada, que puede tener

en las estanterías de su sección de informática algún libro que le proporcione la respuesta. Las principales fuentes de información específica para el Amstrad son, por supuesto, las publicaciones de Amsoft, que cubren casi todos los aspectos de las máquinas en gran detalle. Yo no mantengo correspondencia sobre los proyectos, salvo en circunstancias excepcionales.

Diríjase el principio de este libro para obtener detalles sobre los kits de componentes de los proyectos y también para saber cómo puede comprar el juego completo del 'software' en disco, que le ahorrará una gran cantidad de tiempo.

CAPITULO CUATRO
Usando la ROM del Amstrad
Control 'Hardware' desde el BASIC

Cuando use el BASIC del Amstrad, encontrará muchas facilidades que no están disponibles en máquinas menores. Algunos fabricantes no le permiten controlar el 'hardware' u otros ordenadores. El Amstrad si se lo permite. A continuación listamos las facilidades del BASIC aplicables para el control directo del 'hardware' o que proporcionan información sobre el 'hardware', así como algunos ejemplos de los menos claros. Los ejemplos se dan en orden alfabético:

BORDER
Este comando le dice a la matriz de puertas de vídeo (ver capítulo 1) de qué color debe poner los puntos del borde de la pantalla.

CAT
Le dice al 'software' de cinta o disco que los lea y produzca un catálogo en la pantalla.

CLS/CLG
Estos dos comandos hacen que se llene toda la memoria de pantalla con el color actual de fondo (o PAPER).

DI
Esta no es una sentencia de control de 'hardware', pero se incluye aquí para resaltarlo. No se le debe confundir con la instrucción del Z80A del mismo nombre. Si se desactivaran las interrupciones, el teclado se quedaría inoperativo, ya que se analiza durante la rutina de interrupción que se ejecuta siempre, detrás de los que parece que está haciendo la máquina.

EOF
Muy útil en el manejo de ficheros. Comprobando la variable EOF podemos saber si las rutinas de la ROM han obtenido más datos del fichero que se ha abierto mediante la sentencia OPENIN.
EJEMPLO:
```
IF EOF=0 THEN PRINT "NO HAY MAS DATOS" ELSE PRINT "HAY MAS DATOS DISPONIBLES"
```

FRE
Muestra la cantidad de memoria libre. La forma FRE("") fuerza la limpieza de memoria por el BASIC.

HIMEM
Es una variable que nos da la dirección de memoria más lata que puede usar el BASIC, dejando el resto libre para otros usos como la carga de programas en código máquina.

INP (Número)
Lee la puerta de entrada correspondiente al número.
EJEMPLO:
```
IF (INP(&F500) AND &40)=0 THEN PRINT "IMPRESORA PREPARADA"
```

MEMORY
Pone en el parámetro HIMEM el valor requerido.
EJEMPLO:
```
MEMORY 15000
```

MODE x
Pone el modo de pantalla del chip 6845 controlador VDU.
EJEMPLO:
```
MODE 1
```

OPENIN/OPENOUT
OPENIN lee un bloque completo de información desde el disco o la cinta. OPENOUT crea un bloque de datos destinados a ser escrito sobre disco o cinta.

OUT
Proporciona una vía para mandar datos a una dirección de E/S, (es opuesto a INP).

PEEK/POKE
Permite leer directamente un octeto de memoria o escribir un valor dentro de un octeto de memoria.
EJEMPLO:
```
POKE 40000,6: PRINT "LA DIRECCION 40000 CONTIENE ";PEEK(40000)
```

POS
Esta función nos dice en que columna del canal entre paréntesis está localizado el cursor o la cabeza de impresión. Si es diferente de la posición real, el programa debe corregirlo, lo veremos más adelante en este capítulo.

SAVE
Hace que el BASIC llame a las rutinas de la ROM que escriben datos en disco o cinta.

VPOS
Es similar a POS, excepto en que esta nos da la posición vertical dentro de la pantalla o página de la impresora donde está localizado el cursor o la cabeza de impresión.

WIDTH
Es un comando que le dice a la ROM cuantas columnas puede imprimir la impresora conectada al canal 8, suele ser 80 o 132. (ver más adelante).

Estas son las palabras clave del BASIC que más afectan al 'hardware' de las máquinas Amstrad. Veamos ahora algunos puntos sobre cómo usa realmente el BASIC el 'hardware', y como puede causar algunos problemas la interacción entre el 'hardware' y el 'software'.

INTERACCIONES ENTRE EL 'HARDWARE' Y EL 'SOFTWARE'.

Empezaremos con la causa principal de confusión - la impresora. Si usted teclea el comando directo:

```
PRINT POS(#8)
```

aparecerá el número de columna en el que piensa la ROM que está la

cabeza de impresión en ese momento. Si no coincide con la posición real, la ROM insertará saltos de línea y retornos de carro en su informe a destiempo. Sería mejor que, antes de intentar imprimir algo, tecleara:

```
WIDTH xx: PRINT #8
```

Donde xx es el número de columnas de ancho de la impresora. Esto le asegura que coinciden la ROM y la impresora. Si intenta imprimir algo sin usar el comando WIDTH, nada más encender la impresora, ésta puede aparecer mal alineada - aún peor, si está usando una impresora de 80 columnas la impresora puede imprimir fuera del papel.

¡LAS INTERRUPCIONES SON NUESTRAS AMIGAS!

Como ya hemos dicho anteriormente en este capítulo, el Z80A del CPC está recibiendo, reconociendo y tomando acciones de continuos requerimientos de interrupciones. Desde el BASIC no podemos darnos cuenta de este tipo de cosas. Piense en el 'hardware' del teclado, que se ha tratado en el capítulo uno, ¿Son los circuitos asociados a él inteligentes en alguna forma? Otra función es la de mantener la hora, a la que se accede mediante la variable TIME desde el intérprete BASIC, ¿cómo se mantiene la cuenta de la hora si no hay un reloj dentro del CPC?.

El iniciador de todas estas funciones es el generador de interrupciones. Estas se originan desde la matriz de puertas de vídeo y suceden cada 1/300 de segundo. La rutina que sirve estas interrupciones - llamada interrupción de tiempo - puede detectar si ésta ha sido generada por un dispositivo de usuario conectado, o si es una generada internamente. Durante la rutina de interrupción se realizan ciertas acciones, que están relacionadas con el contador de tiempo del sistema, y a las que se accede mediante las palabras clave del BASIC, AFTER y EVERY. La rutina de interrupción de 1/300 de segundo mira si debe actualizar el reloj, o ha pasado un periodo de tiempo para ejecutar otras funciones de la máquina. Estas funciones incluyen el parpadeo de caracteres, comprobación del generador de sonidos, y análisis del teclado.
De esta forma el Z80A está ejecutando rutinas de interrupción continuamente mientras la máquina permanece encendida, excepto en ciertos casos especiales. Estos casos especiales son las rutinas del cassette y del disco. Como todas las temporizaciones de recuperación de datos del cassette se hacen por programa (para mantener la cantidad de circuitos electrónicos requeridos al mínimo) no permite que haya interrupciones mientras tiene lugar la codificación (escritura) o la decodificación (lectura) de datos.

¿Ha probado usted alguna vez a pulsar la tecla ESC durante el tiempo en que se está leyendo de cinta o disco? Si no lo ha hecho, inténtelo. Verá que no obtiene respuesta, no ocurre el BREAK usual. Esto se debe a que el teclado no se analiza porque están desactivadas las interrupciones. Otro buen experimento que puede hacer es intentar teclear algo en el teclado mientras se ejecuta una operación de E/S. Verá que la mayor parte de lo que ha tecleado no es reconocido por la máquina.

Otra implicación del hecho de no permitir las interrupciones durante las operaciones de E/S es que se detiene el funcionamiento del re-

loj. Como se especifica en el manual del usuario del CPC, esto significa que los programas BASIC que usan la variable TIME, al mismo tiempo que usan ficheros de datos, no deben esperar que la función TIME tenga exactitud en la especificación del tiempo trascurrido desde que se encendió la máquina.

CONCLUSIÓN

El BASIC suministrado con las máquinas CPC es bastante bueno y le permite acceder al 'hardware', a la memoria, al espacio de E/S, etc. También es bastante bueno ocultándose. No puede hacer PEEK de la ROM del BASIC, ya que todas las referencias al área de memoria donde reside son redireccionadas a la RAM que ocupa el mismo espacio de direcciones. Esto significa que desde el BASIC no se puede listar la ROM del BASIC.

El siguiente capítulo es una introducción a los conceptos y técnicas usadas en un lenguaje que le permite, como programador, conectar el 'hardware' interno con sus propias unidades externas.

CAPITULO CINCO
Desarrollando Programas en Lenguaje Ensamblador

La programación en BASIC le puede volver algo vago. El BASIC está destinado a escribir programas simples en el teclado, con poca o ninguna preparación. Es casi siempre cierto que el programa escrito de esta forma es bastante menos eficiente que si hemos hecho diagramas de flujo antes de codificar el programa BASIC. La ventaja es que si usted se equivoca, el BASIC se lo dirá por medio de un inteligente mensaje de error.

Sin embargo, el BASIC tiene muchos inconvenientes, junto con el de su pereza. El peor es la diferencia sustancial de velocidad, si lo comparamos con otros lenguajes, especialmente con el código máquina. Un programa en código máquina puede ejecutar una tarea que funcione 25 veces más rápida que un programa BASIC que haga lo mismo. Muchas versiones de BASIC (no el Locomotive BASIC del CPC) limitan la cantidad de control directo que pueden ejercer los programas sobre el 'hardware' en el que se ejecutan. Y finalmente, los programas BASIC son poco eficientes. Por estas razones, después de dominar un lenguaje como el BASIC, muchos programadores pasan al más tedioso, y menos seguro proceso de escribir programas en ensamblador. Ellos han descubierto que la atención al detalle que deben ejercitar, junto con la planificación previa más extensa, y los resultados finales más eficientes, les producen mayores satisfacciones una vez que se ha completado el programa.

Ya sabe que el microprocesador Z80A, que es el cerebro de su CPC, no entiende directamente las sentencias como PRINT o READ o GOTO. Solo entiende números, e interpreta ciertos números como instrucciones. Usted puede escribir estos números en el formato que usted desee, ya sea en hexadecimal, octal, decimal o en binario. El Z80A solo necesita ver el código correcto, así como reconocer las máscaras binarias que forman las instrucciones, para ser feliz.

En el capítulo dos hemos visto el juego de instrucciones del Z80A, y en este capítulo veremos como puede poner esas instrucciones a trabajar. Idealmente debe poseer un paquete ensamblador como el DEVPAC (Disponible en los comercios de microordenadores). Si solamente quiere copiar los programas presentados en este libro, se le dará una forma de usar las sentencias BASIC, DATA y POKE para crear en memoria programas en código máquina ejecutable.

¿QUÉ ES UN PROGRAMA EN ENSAMBLADOR?

Si tuvo alguno de los primeros ordenadores personales, como el Science o el Cambridge MK14 o el Acorn original (circa 1977-1978), sabrá que había pocas oportunidades de programarlos en otro lenguaje que no fuera el código máquina. Esto significa que usted tenía un li-

bro de instrucciones, que incluía una lista de códigos de operación (opcodes) que entendía el microprocesador. El método de entrada era un pequeño teclado con los dígitos hexadecimales y algunas teclas de comandos. La pantalla era una matriz de ocho dígitos LED al estilo de una calculadora. Usted introducía el programa pulsando una secuencia de teclas como: "MEM" "F" "E" "F" "O". Que seleccionaba la dirección de memoria Hex FEFO. Después pulsaba "DEP" "3" "4" para introducir 34 dentro de la posición de memoria. No había BASIC ni teclado convencional. La programación en esta forma consumía mucho tiempo y producía aburrimiento. Todo lo tenía que hacer usted, códigos de operación, valores de desplazamiento para los saltos, y todo lo demás. Después, cuando se necesitaban hacer inserciones y borrados en el programa, debían volver a calcularse manualmente todos los desplazamientos.

Gracias a los teclados convencionales (QWERTY) y las pantallas de texto, las generaciones posteriores de ordenadores personales, como la máquina CPC, han hecho posible el uso de programas ensambladores. Estos aceptan desde el teclado el cuerpo del programa que representa las instrucciones de código máquina en forma de texto. La parte más potente viene cuando se ejecuta un comando ensamblador. Cuando esto sucede, el ensamblador entra en acción y produce un programa en código máquina que puede ser ejecutado por el microprocesador. Durante el ensamblaje, los mensajes de error le dicen si ha cometido errores sintácticos, aunque no suele dar los errores de lógica, por ejemplo un bucle sin salida. Ignore los mensajes de error bajo su responsabilidad. Recuerde que cuando se ha completado un ensamblaje, y pasa a ejecutar el programa en código máquina que se ha creado, usted no puede pulsar la tecla ESC para salir de él. El programa se debe ejecutar correctamente, o la máquina fallará con toda seguridad. Si lo hace tendrá que apagarla y cargar todo de nuevo - el ensamblador y todo lo demás - desde cinta o disco. Los ensambladores tienen también facilidades de edición, como un pequeño procesador de textos. El DEVPAC tiene también un pequeño programa de depuración, que veremos más adelante.

Vamos a ver los distintos términos que vamos a usar: el ENSAMBLADOR toma el PROGRAMA en LEGUAJE ENSAMBLADOR y produce el CODIGO MAQUINA que lo implementará (Otro nombre que puede oír usted es CODIGO OBJETO. Este es otro nombre del programa en código máquina - porque es el objeto del programa en ensamblador). El informe producido durante el proceso de ensamblaje se llama LISTADO del ENSAMBLAJE.

ELEMENTOS NO CÓDIGO MÁQUINA DEL LENGUAJE ENSAMBLADOR

Así como nemotécnicos de código máquina, usted encontrará otras cosas en los listados de ensamblador. Los más obvios son los comentarios. Siempre van precedidos por un punto y coma. Otra facilidad familiar desde el BASIC es que cada línea de ensamblador tiene un número de línea. Veamos un ejemplo de una línea que carga el registro "A" con el contenido del registro "D":

```
100       LD   A,D      ; Carga el registro A desde el registro D

Num. línea   Nemotécnico   Comentarios
```

En el ensamblador DEVPAC, las líneas pueden ir desde la 1 a la 32767. Los nemotécnicos son las instrucciones legales del Z80A. La sección de comentarios es una breve explicación de lo que hace la línea.

Ya hemos visto como se usan los números de línea, pero no nos sirven como posiciones de la forma en que se usan en un programa BASIC. No se puede ejecutar una instrucción GOTO 100 en Z80A para ir a la línea 100. Los números de línea se usan durante el ensamblaje, o durante la edición para identificar la línea que contiene un error, o qué línea quiere usted editar. Si quiere ejecutar, o referenciar un punto del programa más de una vez, debe darle una etiqueta a la línea. Una etiqueta es un nombre de hasta seis caracteres seguido por dos puntos. Por ejemplo, podemos etiquetar la línea que acabamos de usar como "COPYDA".

```
100 COPYDA:  LD  A,D   ; Copia el registro D al registro A
      |
   Etiqueta
```

Esto le permitirá el uso de una sentencia de "Salto Relativo", que le permitirá volver a este punto en cualquier parte del programa, por medio de una línea como esta:

```
900          JR  COPYDA  ; Salta al punto donde A es cargado desde D
```

Otro grupo de elementos en el listado, que no son nemotécnicos de código máquina, son las directrices. Una directriz es un comando que se incluye en el listado para decirle al programa ensamblador que ejecute ciertas acciones durante el ensamblaje. Estas son cosas como decirle que imprima en hexadecimal todos los valores del listado del ensamblaje, o que reserve algunos octetos de memoria para almacenar los valores requeridos por el programa. Encontraremos muchas de estas directrices a medida que veamos listados de programas en este capítulo, y se irán explicando a medida que aparezcan.

El tipo final de elemento en un listado, que no es código máquina, es una línea en blanco. Esta se puede usar para hacer buen efecto e incrementar la legibilidad del programa, separando las diferentes rutinas, y haciendo más fácil de localizar los puntos específicos del programa.

Para que pueda experimentar, los programas en ensamblador de este capítulo son los escritos para las subrutinas en código máquina que aparecen listados en sentencias DATA o en vuelcos hexadecimales en el capítulo tres. Teniendo los listados en ensamblador podrá comprenderlos, ajustarlos o reescribirlos, según sus necesidades.

VAMOS ADELANTE

Si ya sabe como usar el DEVPAC en su CPC, o si tiene la versión en disco del DEVPAC, sáltese esta sección.

Es una buena idea tener el ensamblador y el programa de depuración, GENA3 y MONA3 respectivamente, cargados simultáneamente. Esto nos permitirá probar los programas según los desarrollamos. La mayoría de la información requerida para usar los dos programas está contenida en el manual que viene con ellos, yo, y otros, hemos descubierto que

las instrucciones para cargarlos juntos no son muy exactas, así que vamos a explicar la secuencia para poder hacerlo. Primero cargue la cara dos del cassette DEVPAC. Después teclee: *RUN ""* y pulse la tecla PLAY de su 'datacorder'. Después de 20 segundos el programa MONALOADER se ejecutará y le preguntará *"LOAD ADDRESS?"* (¿DIRECCION DE CARGA?). Si quiere cargar ambos módulos deberá especificar 30000. Entonces MONA3 se cargará y, cuando se haya completado la carga, se ejecutará. Lo primero que hace es imprimir su panel frontal sobre la pantalla. Teclee ahora CTRL/X (CTRL y X simultáneamente) para volver al BASIC, puede que tenga que pulsar un par de veces ENTER para obtener el mensaje "READY". Ahora déle la vuelta al cassette para ponerlo en la cara uno, rebobínelo y pulse PLAY. Teclee de nuevo: *RUN ""*. Se cargará GENLOADER que cuando se ejecute le preguntará *"LOAD ADDRESS?"* Aquí le debe responder 1000. Entonces se cargará GENA3, imprimiendo su menú de comandos cuando esté preparado para funcionar. Teclee B y ENTER para volver al BASIC, de nuevo deberá pulsar un par de veces ENTER para obtener el mensaje de "READY". Teclee ahora *CALL 1004*. Esto le introducirá de nuevo en GENA3, pero salga otra vez tecleando el comando B. Esta vez irá directamente al BASIC. Finalmente teclee *CALL 30002*. Esto le introducirá en MONA3. Desde ahora, si está ejecutando MONA3 y quiere ir a GENA3 teclee simplemente CTRL/J, si está en GENA3 y quiere ir a MONA3 teclee J y ENTER. De esta forma no necesita pasar por el BASIC.

Mientras usa GENA3 tiene la opción de especificar la dirección donde quiere que escriba el código máquina la fase de ensamblaje. Esto se hace mediante la directriz *ORG* (Organizar). GENA3 le lanzará un mensaje de error *"BAD ORG!"* si el ORG que usted ha especificado hace que se escriba el código sobre la representación textual del programa que está escribiendo (llamada fichero de texto), o sobre el programa GENA3. Tenga cuidado al especificar el ORG si éste escribe sobre el área de memoria ocupada por MONA3, GENA3 no le indicará nada en esta caso.

No es mi intención duplicar el contenido del manual del DEVPAC de Hisoft (Hisoft debería haber dado algún ejemplo más) así que ahora vamos a pasar a ver algunos programas prácticos, que se explicarán paso a paso.

ESCRIBIENDO PROGRAMA EN LENGUAJE ENSAMBLADOR

En este punto debo llamar su atención sobre un libro llamado SOFT 158. Este libro es una publicación de Amsoft que da detalles sobre las rutinas de la ROM del CPC. Deberá tener disponibles este libro y alguno de Z80 - de los que damos detalles en el apéndice 2 - si intenta escribir muchos programas en su CPC. SOFT 158 nos da detalles no solamente sobre la asignación de direcciones de E/S y sobre cómo usar las subrutinas de la parte baja de la ROM, sino que también da detalles sobre las E/S. SOFT 158 es caro, pero si quiere ser programador de bajo nivel del CPC, lo necesitará, y será una buena inversión.

Cuando se disponga a escribir programas en lenguaje ensamblador, aunque sea un programa pequeño, es muy importante que empiece con una idea clara de qué es lo que quiere hacer y de cómo lo quiere hacer. Esto implica el uso de diagramas de flujo. Un diagrama de flujo es u-

na representación sobre el papel de las acciones y decisiones que ocurren en el programa, empezando por los pasos de iniciación, y siguiendo a través de todos los caminos posibles para terminar el programa. Los principales símbolos que se usan en los diagramas de flujo están representados en la figura 5.1. Dentro de cada uno hay ejemplos del tipo de información que puede encontrar en un diagrama real. La figura 5.2 nos muestra un diagrama de flujo para decidir el proceso que debe seguir el lector para leer este capítulo.

Cuando dibuje diagramas de flujo le será, fácil localizar los errores de lógica en su aproximación a la implementación del programa. Considere el diagrama de la figura 5.3. En un vistazo podemos ver que el registro "A" nunca llegará a cero, ya que se está cargando siempre con diez. Este es un ejemplo simple, pero cuando escriba programas más complejos detectará, y prevendrá los errores mucho más fácilmente sobre el diagrama de flujo que cuando introduzca el programa desde el teclado.

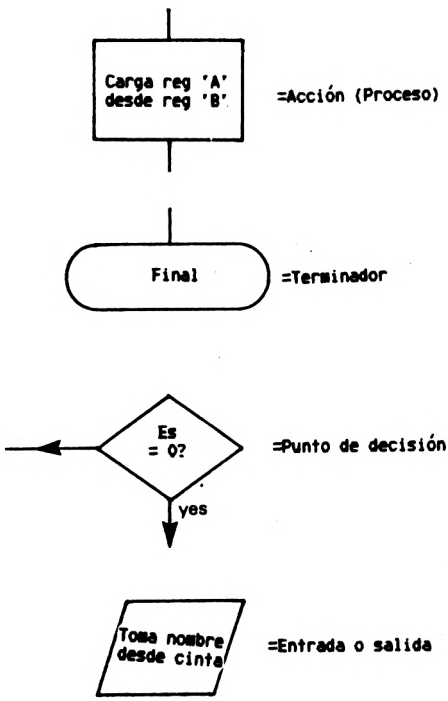
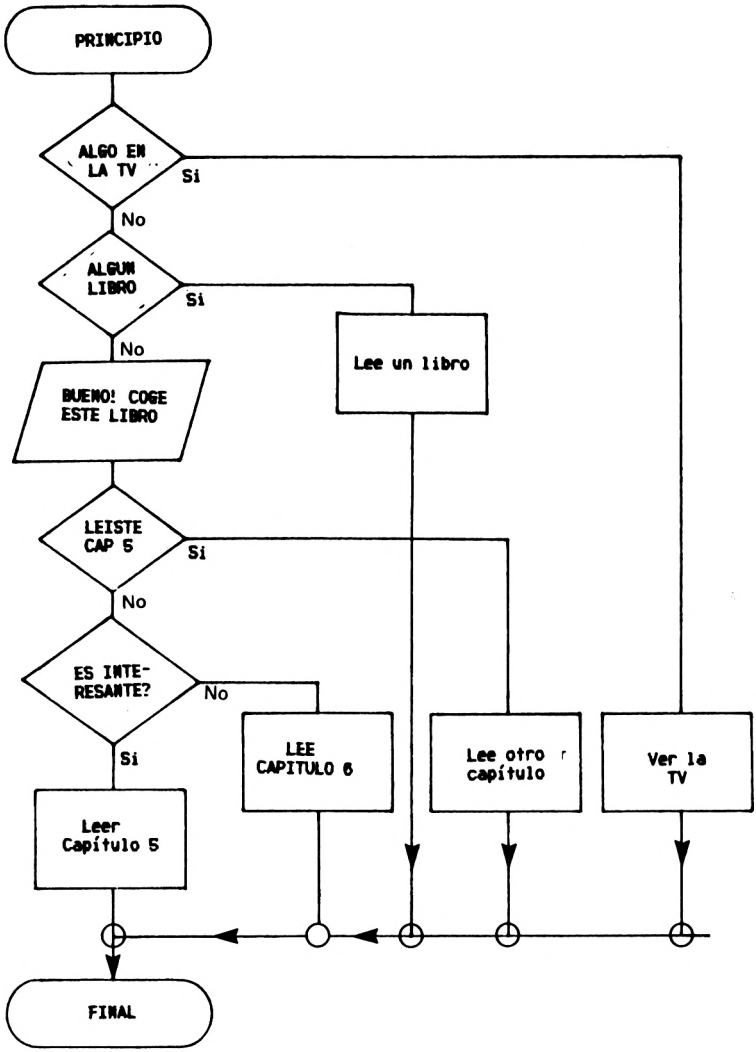


Fig 5.1 Símbolos de Diagrama de Flujo

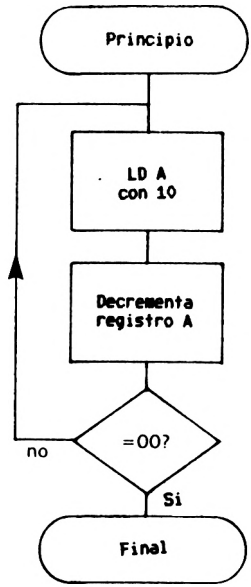
Fig 5.2 Diagrama de decisión del lector



PROGRAMAS DEL PROYECTO CINCO

En el proyecto cinco del capítulo tres vimos cómo se usa una puerta de entrada de ocho bits como registro de sensores. Ahora vamos a desarrollar la parte de 'software' del proyecto. Vamos a conectar ocho

Fig 5.3 Error típico - Detectado en el diagrama de flujo



pulsadores a la puerta, uno por cada bit. Puede montar los pulsadores en una caja conectada con un cable largo al CPC. Los pulsadores deben ser de disparo rápido. En el ejemplo de aplicación contábamos cuantas especies de pájaros diferentes veíamos, para ello debereamos llevar un cable largo desde el CPC hasta un lugar oculto. El programa que vamos a ver se puede acoplar fácilmente para cualquier aplicación de cuenta, ya sean coches, trenes, estudios de tráfico o cualquier otra cosa.

La función principal del programa es contar el número de pulsaciones de cada pulsador, y mostrar los resultados sobre la pantalla. Analizamos repetidamente la puerta y contamos cuantas veces se pulsa cada pulsador. Los contadores deben ser capaces de contar desde 0 a 9999. La versión final proporciona algunas otras facilidades.

Si se pulsa SHIFT/3 (esto es, el símbolo #), el programa para el análisis de los pulsadores y escribe los valores contenidos en los contadores a cinta o disco. La distribución de los contadores en el fichero de datos se detallará más adelante, pero su propósito es permitir escribir un programa en BASIC que produzca informes con porcentajes, totales en un periodo dado, etc. Puede detener el programa pulsando CTRL junto con la flecha hacia arriba que hay sobre la tecla COPY.

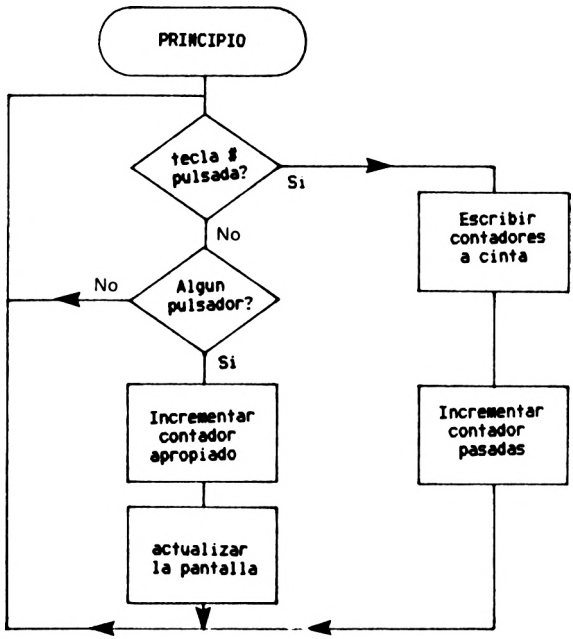
Ahora que hemos establecido los requerimientos, podemos especificar

que necesitamos los siguientes módulos:

1) Una sección de iniciación.
2) Una sección de análisis de los pulsadores
3) Una sección de actualización de contadores
4) Una sección de actualización de la pantalla
5) Una sección de E/S

Después de considerar lo que debe hacer cada una de estas secciones, podemos seguir adelante escribiendo los diagramas de flujo detallados, y proceder después a escribir los listados de los programas. El diagrama de flujo general se puede ver en la figura 5.4.

Figura 5.4 Diagrama general del programa de registro de datos



La sección de iniciación debe: inicializar PPI #2 para poner la puerta B como puerta de entrada. Debe haber un contador de pasadas, para contar el número de muestras que hemos tomado, de forma que al principio debe estar a cero. El siguiente trabajo es poner a cero todos los contadores (excepto el de pasadas), crear después la pantalla usando las rutina de la ROM para limpiar la pantalla, e imprimir la

Fig 5.5 Diagrama de "DLINIT"

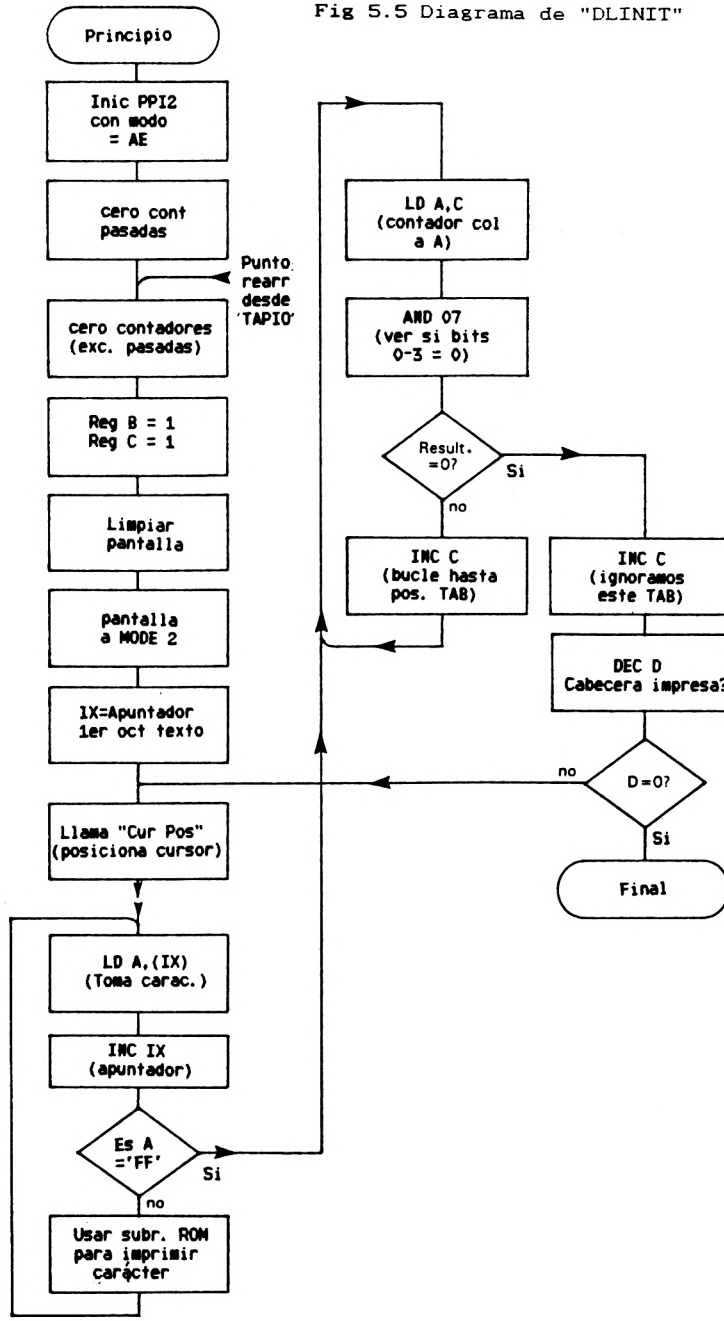


Fig 5.6 Diagrama de "SWSCAN"

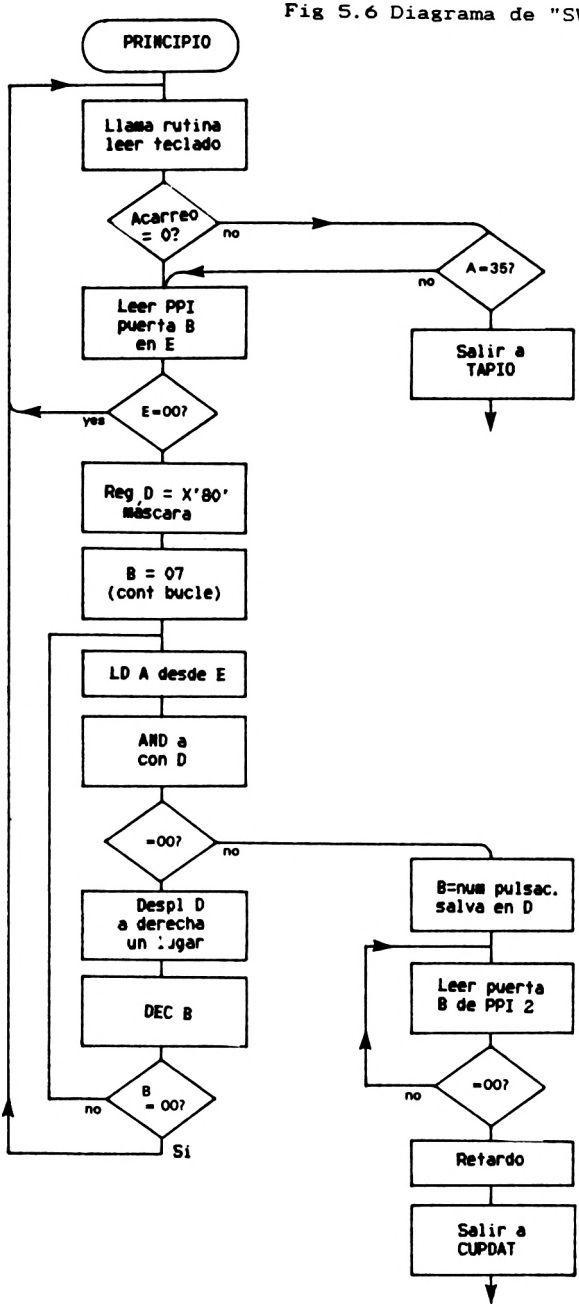


Fig 5.7 Diagrama de "CUPDAT"

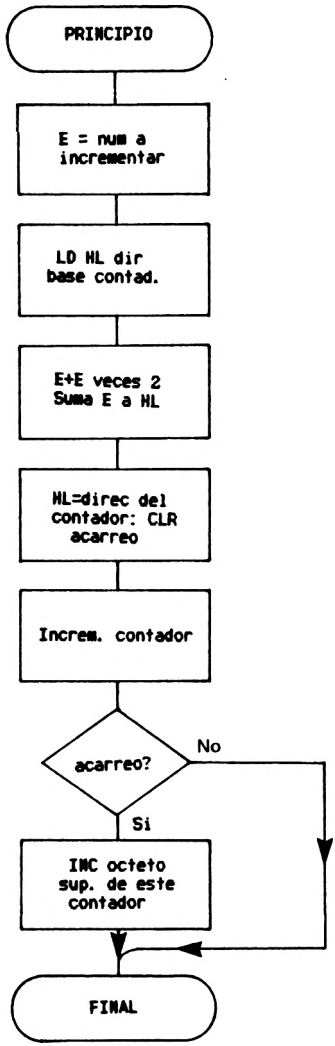
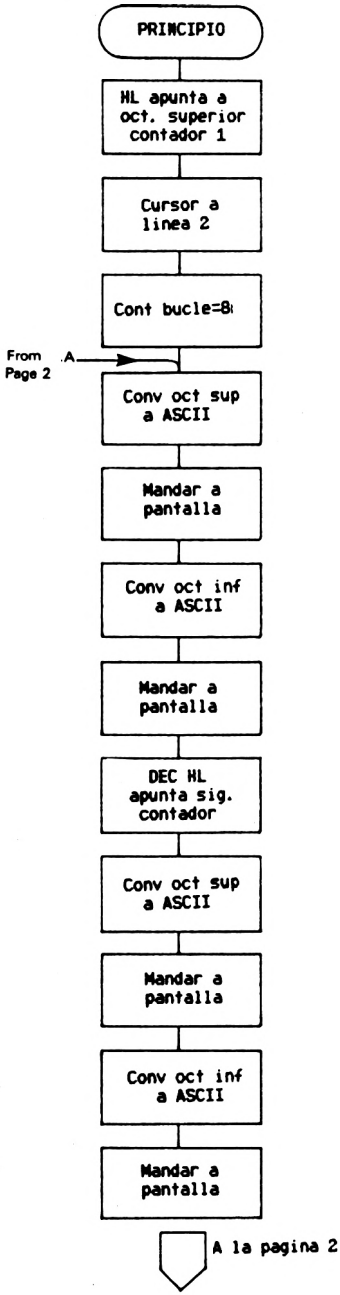


Fig 5.8 Diagrama de "DISPUP" (1 de 2)



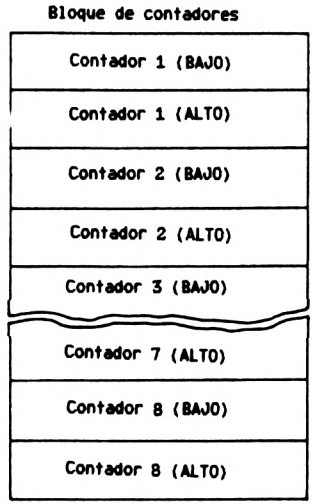
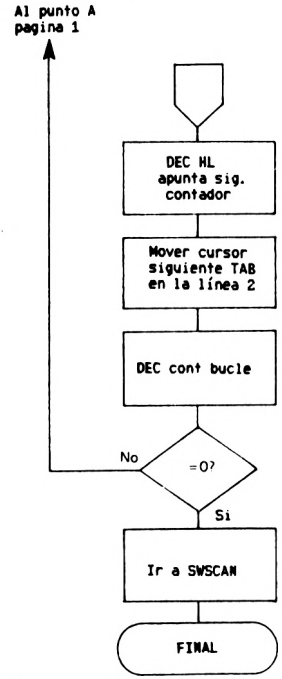


Fig 5.8 Diagrama de "DISPUP" (2 de 2)



cabecera de cada contador; pasando control a la sección de análisis de pulsadores. Debe haber un punto de reentrada en la sección de inicialización para permitir que se rearranque todo menos el contador de pasadas. Este se usa después de escribir todos los datos a un fichero. La figura 5.5 muestra un diagrama de flujo de estas acciones.

La sección de análisis de pulsadores debe: Llamar a la rutina de la ROM para ver si el usuario a tecleado alguna de las combinaciones de teclas, si se ha tecleado # debe pasar control a la sección de manejo de cinta o disco, si se ha pulsado CTRL y la flecha hacia arriba, debe retornar - normalmente al BASIC. Si no se han tecleado caracteres que requieran una acción en el teclado del CPC, el programa debe leer el estado de la puerta B en PPI #2, y comprobar después cada bit por turno para ver si fue pulsado el pulsador conectado a él, si lo fue, debe esperar a que se suelte. Cuando se suelta el pulsador se ejecuta un reardo para evitar los rebotes de la mecánica del pulsador. Pasa control a la sección de actualización de los contadores, mandándole el número del pulsador que se ha pulsador. La figura 5.6 es la representación de estos pasos en forma de
diagrama de flujo.

La sección de actualización de los contadores incrementará uno de los ocho contadores, el contador que se incrementa es el indicado por el valor recibido del analizador de pulsadores. La cuenta se va a llevar en decimal - no en hexadecimal. Cuando se ha hecho, se pasa

control a la sección de actualización de la pantalla. La figura 5.7 muestra el diagrama de flujo de la sección de actualización de contadores.

La sección de actualización de la pantalla debe convertir los valores BCD (decimal codificado en binario) de los contadores, en códigos ASCII, que se puedan mandar a la ROM del CPC para que los imprima en su lugar de la pantalla. Después de hacer esto, se devuelve control al analizador de pulsadores. El diagrama de esta sección está en la figura 5.8.

La sección de E/S prepara una llamada a las rutinas de la ROM que escribirán los valores en un fichero, siguiendo las convenciones del CPC en cuanto a apertura y cierre de ficheros. Después de escribir el fichero, se incrementa el contador de pasadas. Ahora se pasa control al punto de reentrada de la sección de iniciación - preparándose para otra pasada. El diagrama de flujo de esta parte se ve en la figura 5.9.

DESARROLLANDO PROGRAMAS DESDE LOS DIAGRAMAS DE FLUJO

Hemos visto los diagramas de flujo de la aplicación del registrador de datos, ahora vamos a pasar a la fase final de escritura de nuestro programa ejemplo, que es convertir los diagramas de flujo en programas ejecutables.

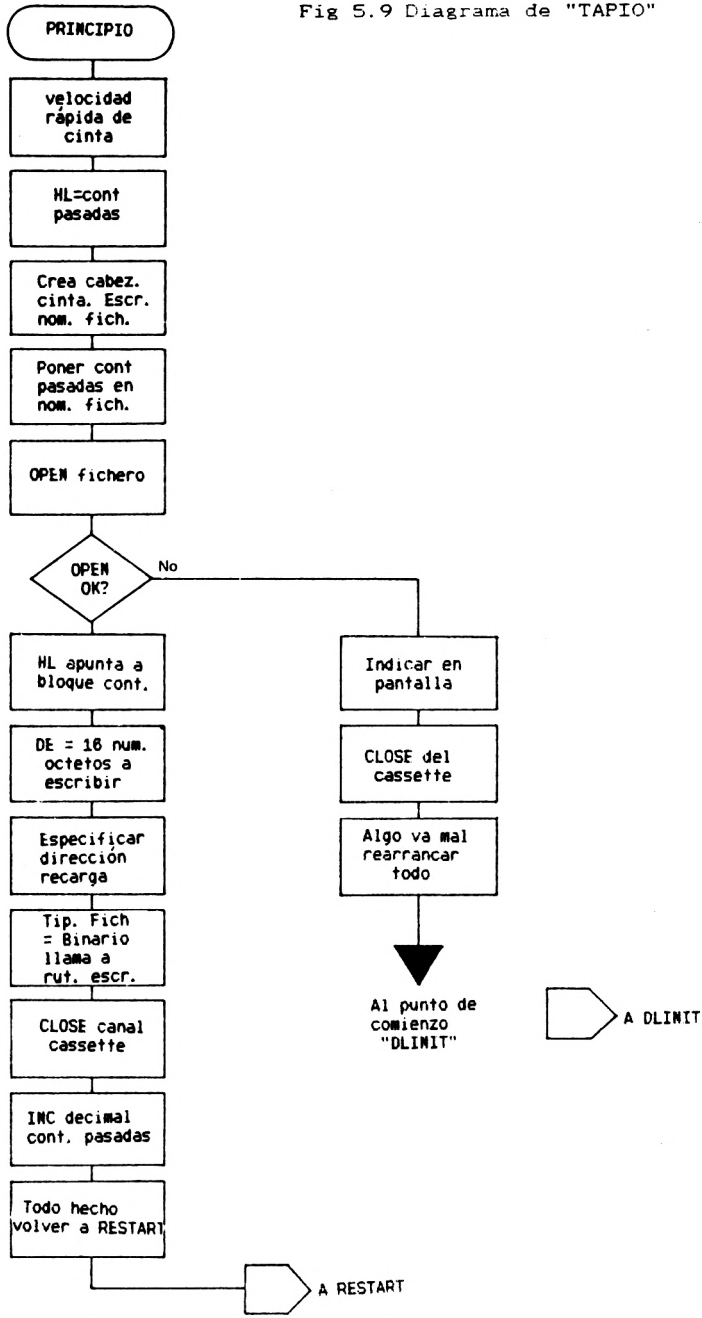
En las explicaciones detalladas de los listados de los módulos, que vienen a continuación, cuando no nos hemos referido anteriormente a alguna instrucción, aparecerá entre paréntesis a un lado el número asignado a ella en la lista de instrucciones del capítulo dos, por ejemplo INC IX (Ref 50). También se explicarán las diferentes directrices y facilidades a medida que las encontremos.

La figura 5.10 es el listado de un programa en lenguaje ensamblador para implementar el módulo de iniciación, llamado DLINIT.

En la cabecera del listado hay algunos comentarios que explican el propósito del programa, y una explicación sucinta. La línea 210 es la primera línea, que no hace nada. Contiene una directriz ORG, que le dice al ensamblador que almacene el código máquina empezando en la dirección Hex 4E20 (20000 decimal). Si por alguna razón quiere que el ensamblador le almacena el código máquina en otra dirección, debe cambiar este valor. Las líneas 220 y 230 realizan dos funciones. Primera, representan un punto que está asociado a las etiquetas PASS y CBASE. Segundo, reservan un octeto como contador PASS y 16 octetos que serán los contadores de los pulsadores. Puede ser útil explicar estas dos directrices con más detalle.

La directriz DEFB (Definir Byte - Octeto) le dice al ensamblador que coloque el valor después del DEFB (cero en este caso) en la siguiente dirección de memoria a la que apunta el contador de posiciones. El contador de posiciones del ensamblador apunta a la dirección de memoria donde se va a colocar el siguiente código máquina generado, en este caso, se colocará el valor indicado en lugar de un código máquina.

Fig 5.9 Diagrama de "TAPIO"



```
10
20 ;       ** Registro de Datos **
30
40 ;
50 ;
60 ;
70 ;
80 ;   El programa está dividido en los siguientes módulos:
90 ;
100
110 ;         1) "DLINIT" - Inicializa
120 ;         2) "SWSCAN" - Análiza los pulsadores
130 ;         3) "CPUDAT" - Actualiza los contadores
140 ;         4) "DISPUP" - Actualiza los contadores en pantalla
150 ;         5) "TAPIO"  - Sección de E/S a cinta
160
170 ; ** Además incorpora algunas
180 ; ** subrutinas más.
190
200
210           ORG  20000
220 PASS:     DEFB 0            ; Contador de pasadas
230 CBASE:    DEFS 16           ; 16 octetos para los contadores
240 DATLOG:   ENT  $            ; Adelante!
250
260 DLINIT:   LD   BC,#F8F7
270           LD   A,#AE  ; Cargar octeto de modo en PPI #2
280           OUT  (C),A
290           LD   IX,PASS      ; IX apunta al contador de pasadas
300           LD   (IX),#0      ; Cero al contador de pasadas
310 RESTAR:   LD   D,16   ; Preparar para poner a 0 los 8 contadores
320           LD   IX,CBASE     ; IX apunta a los contadores
330 INIT00:   LD   (IX),#0      ; Poner el contador a cero
340           INC  IX           ; Incrementar IX
350           DEC  D
360           JR   NZ,INIT00    ; Repetir si hay más contadores
370           LD   A,#04  ; Poner MODE 2 de pantalla
380           CALL #BB5A  ; Usar la ROM para sacar el carácter
390           LD   A,#2   ; MODE 2, por favor
400           CALL #BB5A
410           LD   IX,LABTXT    ; IX apunta a los textos de las etiquetas
420           LD   D,8          ; D nos da 8 bucles
430           LD   B,#1   ; Usar B para número línea de pantalla
440           LD   C,#1   ; Usar C para número columna de pantalla
450 INIT01:   CALL CURPOS ; Llama a subrutina de posición del cursor
460 INIT02:   LD   A,(IX) ; Tomar un octeto de texto en A
470           INC  IX           ; Incrementar IX
480           CP   #FF          ; Ver si es fin de texto
490           JR   Z,INIT03     ; Si, saltar
500           CALL #BB5A  ; No, mandar el carácter
510           JR   INIT02 ; y repetir de nuevo
520 INIT03:   LD   A,C          ; Cargar en A el contador de columnas
530           AND  #07          ; Ver si estamos en un punto de tabulador
540           JR   Z,INIT04     ; Si, saltar
550           INC  C            ; No, incrementar C
560           JR   INIT03 ; Repetir unas cuantas veces
570 INIT04:   INC  C            ; Mover C al TAB para la próxima vez
580           DEC  D            ; Ver si se han sacado las 8 cabeceras
590           JR   NZ,INIT01    ; No, saltar
```

```
600             JP   SWSCAN ; Si, terminado. pasar a SWSCAN
610
620
630  CURPOS:    LD   A,31   ; ** Subrutina de posicionar cursor
640             CALL #BB5A  ; C=Num col  B=línea pantalla
650             LD   A,C        ; Mandar columna
660             CALL #BB5A
670             LD   A,B        ; Mandar línea
680             CALL #BB5A
690             RET             ; Terminado, volver
700
710  ; ** Los siguientes son solamente ejemplos. hasta 7 caracteres máximo
720  LABTXT:    DEFM "GORRION"
730             DEFB #FF
740             DEFM "LECH"
750             DEFB #FF
760             DEFM "ABADE"
770             DEFB #FF
780             DEFM "CUERV"
790             DEFB #FF
800             DEFM "HALC"
810             DEFB #FF
820             DEFM "PINZON"
830             DEFB #FF
840             DEFM "GAVIOT"
850             DEFB #FF
860             DEFM "JILGU"
870             DEFB #FF
880
```

Fig 5.10

La directriz DEFS (usado en el listado junto a la etiqueta CBASE) le dice al ensamblador que ponga un cero dentro del número de octetos indicados. En el caso de la línea 230 lo que hacemos es reservar 16 octetos de memoria RAM para usarla como contadores. Estos 16 octetos son referidos colectivamente como el bloque de contadores. El primer octeto del bloque es conocido por su etiqueta, CBASE (Dirección del contador BASE). Con dos octetos por cada contador, el bloque de contadores, en relación con la posición CBASE, se distribuye de la siguiente forma:

```
CBASE      = Contador 0, octeto inferior
CBASE +  1 = Contador 0, octeto superior
CBASE +  2 = Contador 1, octeto inferior
CBASE +  3 = Contador 1, octeto superior
CBASE +  4 = Contador 2, octeto inferior
CBASE +  5 = Contador 2, octeto superior
CBASE +  6 = Contador 3, octeto inferior
CBASE +  7 = Contador 3, octeto superior
CBASE +  8 = Contador 4, octeto inferior
CBASE +  9 = Contador 4, octeto superior
CBASE + 10 = Contador 5, octeto inferior
CBASE + 11 = Contador 5, octeto superior
CBASE + 12 = Contador 6, octeto inferior
CBASE + 13 = Contador 6, octeto superior
CBASE + 14 = Contador 7, octeto inferior
CBASE + 15 = Contador 7, octeto superior
```

Si quisiéramos incrementar el contador cinco, incrementaríamos la posición CBASE+10, y si hubiera rebosamiento, incrementaríamos también CBASE+11.
La línea 240 contiene una directriz ENT $. Esta informa el ensamblador que el punto de ENTrada cuando se ejecute el programa, estará aquí.

La línea 260 carga el registro BC con la dirección del registro de control del PPI #2, usando la instrucción LD rp,xxx (REF 64). La línea 270 carga el valor requerido para preparar PPI #2 para la configuración requerida, esto es, con la puerta B como entrada. Esto se hace con la instrucción LD r,xx (REF 65). La línea 280 saca el valor a PPI #2, con una instrucción OUT (C),A (REF 107).

La línea 290 carga la dirección del contador PASS dentro del registro IX, usando la instrucción LD IX,xxxx (REF 89). El ensamblador rellena el campo xxxx con la dirección de la etiqueta PASS. A continuación, en la línea 300, se pone a cero el contador PASS usando LD (IX),#0 (REF 73).

La línea 310 es el punto en el que se debe rearrancar el programa después de escribir a cinta los valores de los contadores. Es también el principio del bucle para poner a cero todas las entradas del bloque de contadores. El registro D se prepara como contador de bucle, con 16 bucles. la línea 320 coloca la dirección base del contador dentro de IX. (Uno de los errores de desarrollo ocurrió aquí. Esta línea incrementaba originalmente IX, pero solo funcionaba si IX contenía ya la dirección de PASS, y este no era el caso más que en la primera pasada). El bucle empieza realmente en la línea 330 que tiene la etiqueta INIT00. En esta línea ponemos a cero el contador al que apunta IX. En la línea 340 usamos INC IX (REF 50) para incrementar el registro IX de forma que apunte al siguiente contador a lo largo del bucle. Luego, en la línea 350, usamos DEC r (REF 26) para decrementar el contador de bucle del registro D. La línea 360 usa JR cc,xx (REF 62) para ver si se ha ejecutado el bucle 16 veces. Si no, la línea 330 lo ejecuta de nuevo. Una vez más el ensamblador rellena el campo xx de la instrucción de Salto Relativo, con el valor requerido para volver a la posición de la etiqueta INIT00.

Ahora ya hemos llegado a la línea 370. Se puede poner el modo de la pantalla del CPC mandando un código 4 a la subrutina de la pantalla, seguido por el número de modo (ver el manual del CPC, capítulo nueve). Nosotros queremos poner el modo 2, así que la línea 370 carga el registro A con cuatro, y en la línea 380 usamos CALL xxxx (REF 17) para acceder a la subrutina de la ROM que manda el valor encontrado en el registro A a la pantalla del CPC. En las líneas 390 y 400 se manda de la misma forma el valor dos a la pantalla. En la línea 410 se carga el registro IX con la dirección de la etiqueta de texto. Este texto corresponde a las etiquetas que aparecerán sobre cada contador representado. Una vez más el campo xxxx es rellenado durante el ensamblaje, con la dirección de la etiqueta LABTXT.

La línea 420 prepara una vez más el registro D como contador de bucle. Las líneas 430 y 440 ponen a uno los registros B y C. Estos valores son los parámetros de la subrutina CURPOS, que encontrará un poco más adelante en las líneas 630 a 690. Lo que hace CURPOS es posicionar el cursor de texto sobre la pantalla, en las coordenadas indicadas por los registros B y C. Esto es equivalente al comando LOCATE del BASIC (ver el manual del usuario del CPC,capítulo ocho).

La línea 450, etiquetada como INIT01, es el principio del bucle que imprime las etiquetas de todos los contadores. La línea 460 usa LD r,(IX) (REF 71) para obtener un octeto del texto, dentro del registro A. Después, la línea 470, incrementa el apuntador IX al texto, y la línea 480 usa CP s (REF 19) para ver si lo que hemos obtenido es un indicador de fin de texto , Hex FF. La línea 490 usa JR cc,xx (REF 61) para saltar, si ha llegado al final de una entrada de texto. La línea 500 se ejecuta si no es final de entrada de texto, y llama a la rutina de la ROM que manda un carácter a la pantalla. La línea 510 vuelve a la línea 460 para repetir el bucle. Si se ha encontrado el final de una entrada de texto, se ejecuta la línea 520, y se carga el registro C, que contiene todavía el contador de columnas de la pantalla, con el contenido del registro A. Las líneas de la 520 a la 560 usan AND s (REF 11) para incrementar el registro C hasta que contenga ceros en los bits 0 a 2. Cuando sucede esto, la columna en la que posiciona el cursor, cuando se usa para llamar a CURPOS, será una parada de tabulación. Entonces en la línea 570, usando INC r (REF 45), se avanza el contenido del registro C en uno, de forma que esté colocado para la próxima vuelta del bucle. Las línea 580 y 590 comprueban si se han impreso en pantalla todas las cabeceras, y vuelve a INIT01 si no.

La línea 600 usa JP xxxx (REF 57) para saltar directamente a la sección SWSCAN del programa, ya que se ha completado la iniciación.

La línea 630 es el principio de la subrutina CURPOS. Es una subrutina para mandar códigos a la VDU (unidad de visualización) para posicionar el cursor en la columna indicada por el valor del registro C, y en la línea de la pantalla indicada por el contenido del registro B. Para hacerlo usa el método de códigos de control, detallado en el manual del usuario. La línea 630 carga el primer código en el registro A, y la línea 640 llama a la rutina de la ROM que manda el código. La línea 650 carga desde el C el número de columna dentro del registro A. La línea 660 llama de nuevo a la rutina que lo manda. La línea 670 carga el número de línea en A, y la línea 680 hace la llamada a la rutina de pantalla. Una vez hecho esto, la instrucción RET (REF 118) de la línea 690 hace que vuelva el control a la rutina que llamó a CURPOS.

La línea 720 es el principio del texto de etiquetas y tiene la etiqueta LABTXT. Tiene también una directriz de ensamblador que no hemos visto aún, se llama DEFM (DEFinir Memoria). Esta directriz requiere un texto entre comillas. Toma los códigos ASCII de los caracteres entre comillas y los coloca secuencialmente en las siguientes posiciones de acuerdo con el apuntador de posiciones del ensamblador. Usando DEFM se pueden crear áreas de memoria intermedia de texto, que son áreas de memoria para copiar a cinta o a pantalla en un cierto punto de la ejecución del programa. NOTA: cuando ensamble el programa solo aparecerá los códigos de los cuatro primeros caracteres, pero el resto se coloca también en memoria. La definición del área de texto se extiende hasta la línea 870, con directrices DEFB insertadas como separadores entre los nombres.

En la figura 5.11 aparece listada la rutina SWSCAN. Las líneas 900 a 980 contienen una breve explicación de la rutina. La línea 1000 tiene la etiqueta SWSCAN, y contiene una llamada a la rutina de la ROM que comprueba si se ha pulsado alguna tecla del teclado del CPC. Si se ha pulsado alguna, nos devolverá en el registro A el código de la tecla pulsada, y el indicador de acarreo a uno. Si no se ha pulsa-

do ninguna tecla el indicador de acarreo estará a cero. La línea 1010 hace que salte si el indicador de acarreo está a cero, si no, la línea 1020 comprueba si el código pasado en el registro A corresponde al carácter #. Si lo es, la línea 1030 salta a la rutina de E/S del cassette TAPIO. Si la tecla pulsada no era # la línea 1040 mira a ver si era CTRL/^. Si lo era, la línea 1050 usa RET cc (REF 119) para retornar al BASIC. La línea 1060 carga la dirección de la puerta B dentro del registro BC, y la línea 1070 usa IN r,(C) (REF 43) para leer el estado de los pulsadores externos. La línea 1080 salta a la línea 1000 si no había pulsadores pulsados.

```
880
890
900 ; ** SWSCAN - analiza los pulsadores para el registrador de datos
910 ; ** Esta rutina se activa después de la inicilización y constituye
920 ; ** la base del programa de registro de datos. Espera hasta que
930 ; ** sea pulsado uno de los pulsadores conectados a la puerta B de
940 ; ** PPI #2, después pasa el número del pulsador a la rutina de
950 ; ** actualización de los contadores. Si se pulsa la tecla
960 ; ** correspondiente al carácter "#", se escriben en cinta los
970 ; ** valores actuales de los contadores y de PASS, por medio de
980 ; ** la rutina TAPIO a la que salta esta rutina
990
1000 SWSCAN:   CALL #BB09  ; Llama a la ROM para ver si
1010           JR   NC,SCAN01   ; se ha pulsado alguna tecla. saltar si no
1020           CP   35          ; ver si era #
1030           JP   Z,TAPIO     ; Si, saltar
1040           CP   #F8         ; ver si es CTRL/FLECHA ARRIBA
1050           RET  Z           ; Si, devolver control
1060 SCAN01:   LD   BC,#F8F5    ; BC apunta a la puerta B de PPI #2
1070           IN   E,(C)  ; Leer puerta B
1080           JR   Z,SWSCAN    ; Si no se ha pulsado un pulsador, repetir
1090           LD   D,#80  ; Cargar una máscara de bits en D
1100           LD   B,8         ; Preparar ocho bucles
1110 SCAN03:   LD   A,E         ; Poner en A el valor de la puerta B
1120           AND  D           ; AND A con D
1130           JR   NZ,SCAN02   ; Saltar si el bit no es cero
1140           SRA  D           ; Desplazar D un bit a la derecha
1150           DJNZ SCAN03 ; Decrem. B y saltar si es cero
1160           JR   SWSCAN ; Puerta B no 0 pero no pulsador, Hmm
1170 SCAN02:   LD   E,B         ; Salvar número pulsador en E
1180 ; ** Nota: el valor en E es el número del pulsador +1
1190 ; ** porque se usa el contador de bucle como número de pulsador
1200           LD   BC,#F8F5    ; Ahora esperamos a que se suelte el pulsador
1210 SCAN04:   IN   A,(C)
1220           AND  D           ; Ver si todavía está pulsado
1230           JR   NZ,SCAN04   ; No, bucle
1240           LD   B,#80  ; Retraso
1250 SCAN05:   DJNZ SCAN05 ; Bucle, 128 veces
1260           DEC  E           ; E=número real del contador
1270           JP   CUPDAT ; Ahora con E= número pulsador
1280 ; pasamos a la rutina de actualización de contadores
1290
1300
```

Fig 5.11

En este punto le sugiero una modificación en el programa que le permitirá una forma de control remoto sobre el CPC.

Cuando llegamos a la línea 1080, tenemos en el registro A el valor que se ha leído de la puerta B. Será muy fácil detectar si se han pulsado al mismo tiempo dos pulsadores particulares de la caja remota, y si se ha hecho, que realice la misma acción que si se hubiera pulsado la tecla #, esto es, escribir el contenido de los contadores a un fichero. Esto le dará la ventaja de que si está usted situado a cierta distancia del su CPC, o no puede salir de su puesto de observación de aves, puede comenzar su sesión cargando el programa, y colocar después un cassette de registro en el 'datacorder', y pulsar la tecla RECORD, o en un 664 poner un disco vacío en la unidad de discos. Cuando esté funcionando, puede pulsar los dos botones al mismo tiempo y el programa salvará los datos automáticamente, sin necesidad de retornar físicamente al ordenador, (Tenga en cuenta que en al CPC464 debe insertar una CALL a CAS NOISY, con A= a un valor distinto de cero para evitar los mensajes del cassette). Después espere unos diez segundos y siga adelante. Si inserta esta facilidad extra, será una buena práctica para un programador de código máquina principiante.

En la línea 1090 ponemos una máscara en el registro D. Una máscara es una configuración determinada de los bits, que se usan para comprobar el estado de los mismos bits en otro registro, verá como se usa cuando avancemos un poco. La línea 1100 prepara ocho bucles, usando B como contador. Estos serán más eficientes si usamos la instrucción DJNZ xx (REF 31). En la línea 1110 cargamos en el registro A el valor que se ha leído desde la puerta B. Luego, en la línea 1120 usamos AND s (REF 11) para ver si el bit para el que hemos puesto la máscara está puesto en el valor que hemos obtenido desde la puerta B, la línea 1130 salta si está puesto. La línea 1140 usa SRA r (REF 141) para desplazar el contenido del registro D un bit hacia la derecha. D contiene el valor de la máscara, que fue puesto inicialmente a Hex 80 (binario 1000 0000). Después, en la línea 1160 se decrementa el registro B, y si no ha llegado a cero nos vamos a otro bucle a SCAN03: si era cero, ha ocurrido lo imposible. Fuimos al bucle porque lo que habíamos leído de la puerta B no era cero, y una vez que hemos comprobado todos los pulsadores, resulta que no hay ninguno pulsado. Como indica mi comentario en esta línea, inténtelo de nuevo.

Un ejemplo nos dará una idea clara de la operación de una máscara. Después de ejecutar el bucle que comienza en la etiqueta SCAN03, por ejemplo dos veces, el registro D debe contener Hex 20 (binario 0010 0000). Usando este método conseguimos una forma de comprobar cada bit de otro registro usando una simple función AND con la máscara como un operando y el registro cuyo contenido queremos comprobar, como el otro. Como se detalla en el apéndice seis, cuando se ejecuta una función AND de ocho bits, en el resultado solamente aparecerán puestos los bits que aparezcan en ambos operandos. Así, si tenemos 0010 0000 en D y 0001 0000 en A, el resultado será 0000 0000. Por tanto solamente ponemos un bit en la máscara, que es el número de bit que queremos comprobar.

En el momento en que llegamos a la línea 1170, el registro B contiene el número del pulsador que se ha pulsado. Mire la línea 1100. Preparamos ocho vueltas del bucle, y en la línea 1150 comprobamos si el contador de bucle ha llegado a cero, si lo ha hecho no vuelve al bucle. Esto significa que el número que hay en el registro B cuando

sale del bucle nunca puede ser cero, pero hemos numerado los pulsadores del 0 al 7. En efecto, el registro B contendrá el número del pulsador más uno. En la línea 1170, por lo tanto, salvamos el número del pulsador en el registro E.

En el bucle de espera que se ejecuta en las líneas 1200 a 1230, la puerta B se relee con la máscara usada en D, para ver si el pulsador permanece aún pulsado. Hacemos un bucle entre las líneas 1200-1220 hasta que se suelta el pulsador. Una vez suelto, ejecutamos otro bucle, que comienza en la línea 1240, y que impide que los rebotes que pudieran ocurrir en los contactos del pulsador nos introduzcan datos erróneos.

Finalmente, en la línea 1260, decrementamos el contenido del registro E para hacerlo corresponder con el número real del pulsador. Ya ha terminado SWSCAN, y se pasa el control a la rutina de actualización de los contadores, CUPDAT, por medio de la sentencia JP de la línea 1270. Mandamos a CUPDAT el número del contador que queremos incrementar dentro del registro E.

CUPDAT comienza en la línea 1300. La figura 5.12 muestra el listado de este módulo del programa de registro de datos. Después de algunos comentarios, la línea 1380 carga cero dentro del registro H, y la línea 1390 usa RL r (REF 122) para rotar un cero dentro del indicador de acarreo del registro F. Después, en la línea 1400, el par de registros HL se carga con la dirección base del bloque de contadores (CBASE). Como ya vimos en la sección anterior sobre SWSCAN, esta rutina toma el número del pulsador que se va a incrementar del registro E. Lo que necesitamos ahora es convertirlo en un desplazamiento dentro del bloque de contadores, de forma que añadiendo este desplazamiento al contenido de HL apuntaremos a la dirección del contador que tenemos que incrementar. Esto lo hacemos desplazando el registro E hacia la izquierda un bit, que es la mejor manera para multiplicar por dos un número pequeño (que no use el bit 7). Esto se hace en la línea 1410. Así que ahora que tenemos en el registro E un desplazamiento usable, ponemos el octeto inferior del registro HL en el registro A, y en la línea 1430 usamos ADD A,r (REF 7) para sumar el desplazamiento, después lo devolvemos al registro L en la línea 1440. Ahora HL apunta directamente al octeto inferior del contador que tenemos que incrementar.

```
1300
1310 ; ** Esta es la rutina de actualización de contadores - CUPDAT
1320 ; ** A la entrada, el registro E contiene el número del contador
1330 ; ** que hay que actualizar. El valor suministrado se ajusta para
1340 ; ** convertirlo en un desplzamiento sobre el bloque de contadores.
1350 ; ** El contador se incrementa en forma decimal usando la
1360 ; ** instrucción DAA del Z80A
1370 ;
1380 CUPDAT:   LD   H,0
1390           RL   H           ; Limpiar el bit de acarreo
1400           LD   HL,CBASE    ; HL apunta al bloque de contadores
1410           RL   E           ; Multiplica E por 2
1420           LD   A,L         ; Copia L sobre A
1430           ADD  A,E         ; Añade el desplazamiento
1440           LD   L,A         ; HL= dirección contador
1450           LD   A,(HL) ; Toma el valor del contador
1460           INC  A           ; Lo incrementa
1470           DAA              ; Ajuste decimal
1480           LD   (HL),A ; Lo vuelve a su lugar
```

```
1490            JR   NC,ALLDUN   ; Ver si hay acarreo
1500            CCF              ; Si, limpiarlo
1510            INC  HL          ; HL=siguinete posición contador
1520            LD   A,(HL) ; Toma el valor del contador
1530            INC  A           ; Lo incrementa
1540            DAA              ; Ajuste decimal
1550            LD   (HL),A ; Lo vuelve a su lugar
1560  ALLDUN:   NOP              ; Ir a DISPUP
```

Fig 5.12

En la línea 1450 metemos el valor del contador dentro del registro A, después, la línea 1460 lo incrementa. Cuando la línea 1470 tiene ajustado a decimal el nuevo valor del contador, mediante la instrucción DAA (REF 25), se coloca de nuevo en la tabla de contadores en la línea 1480. Cuando llegamos a la línea 1490, comprobamos si el octeto inferior tiene rebosamiento (ej pasó de 99 a 00). Si no lo tiene, saltamos a la línea 1560, con la etiqueta ALLDUN. Si hubo acarreo, lo limpiamos con la instrucción CCF (REF 18) que invierte su estado. La siguiente línea, 1510 incrementa el registro HL, usando INC rp (REF 46). HL apunta ahora el octeto superior del contador seleccionado. Después, en las líneas 1510 a 1550 hacemos exactamente los que hicimos con el octeto inferior. En la línea 1560 ya hemos hecho todo lo correspondiente a la rutina CUPDAT.

La línea 1630 es el principio real de la rutina DISPUP, que actualiza la pantalla del CPC. El listado se muestra en la figura 5.13. En la línea 1630 cargamos el par de registros HL con la dirección del octeto superior del primer contador. Si lo pensamos un poco comprenderemos la razón para hacerlo así. Cuando se va a imprimir el valor de un contador en la pantalla, debemos imprimir primero el dígito superior, por ejemplo, 1500 en lugar de 0015, para mil quinientos. En las líneas 1640 a 1660 usamos nuestro viejo amigo CURPOS para posicionar el cursor en la línea dos de la pantalla, y en la columna uno. Una vez hecho esto, ponemos el registro B a ocho porque debemos repetir ocho veces el bucle de la línea 1680. Aquí es donde se pueden complicar algo las cosas.

```
1580
1590  ;  ** Esta rutina actualiza los contadores en pantalla - DISPUP
1600  ;  ** imprime los ocho contadores en la línea 2 de la pantalla
1610
1620
1630  DISPUP:   LD   HL,CBASE+1  ; HL=dir. octeto sup. prim. contador
1640            LD   B,2         ; Línea 2 de pantalla
1650            LD   C,1         ; Columna 1
1660            CALL CURPOS ; Posicionar el cursor
1670            LD   D,8         ; Preparar 8 bucles
1680  DISP01:   LD   A,#30  ; Medio octeto sup. = 30
1690            RLD              ; Primer dígito en A
1700            CALL #BB5A  ; Mandarlo
1710            RLD              ; Mover segundo dígito
1720            CALL #BB5A  ; Mandarlo
1730            RLD              ; Volver a estado original
1740            DEC  HL          ; HL apunta a octeto inferior contador
1750            RLD              ; Mover el tercer dígito
1760            CALL #BB5A  ; mandarlo
```

```
1770            RLD             ; Mover cuarto dígito
1780            CALL #BB5A  ; Mandarlo
1790            RLD             ; Volver a estado original
1800            LD   A,L        ; Incrementar L en tres
1810            ADD  A,#3   ; apuntar al octeto superior contador sig.
1820            LD   L,A        ; devolver valor a L
1830            LD   B,4        ; Cuatro bucles cursor a la derecha
1840            LD   A,9        ; Bucle, cursor en sig. TAB
1850  DISP02:   CALL #BB5A
1860            DJNZ DISP02 ; Bucle cuatro veces
1870            DEC  D          ; Terminados los contadores?
1880            JR   NZ,DISP01  ; No, repetir
1890            JP   SWSCAN ; Volver a analizar pulsadores
1900
```

Fig 5.13

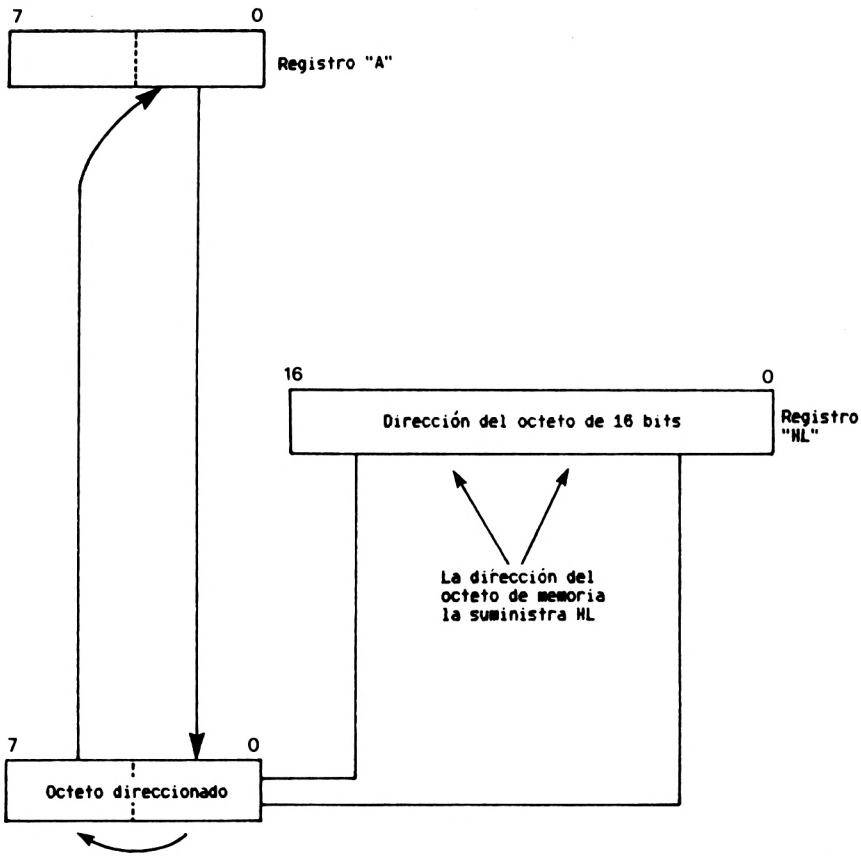
Lo que hacemos en la línea 1680 es convertir cada uno de los cuatro dígitos, contenidos en los dos octetos de cada contador, en ASCII listo para mandar a la rutina de la ROM. La forma de hacerlo es con la instrucción RLD (REF 129). Si no conoce lo que hace la instrucción RLD, eche una ojeada a su descripción en el capítulo dos, y también a la figura 5.14, que la ilustra su acción con un diagrama.

Para empezar, en la línea 1680 se carga el registro A con Hex 30, porque los números ASCII tienen el medio octeto superior con el valor tres. (Ver el apéndice 4 para el diagrama de los valores ASCII). La línea 1690 convierte el contenido del registro A en el código ASCII del dígito más alto, y la línea 1700 llama a la rutina de sacarlo a pantalla. En la línea 1710 se mueve el siguiente dígito al registro A y se saca en la línea 1720, después, la línea 1730 restaura el contenido de la posición a la que apunta el registro HL con otra instrucción RLD. En la línea 1740 se decrementa HL, de forma que apunte al octeto inferior del contador. Desde la línea 1750 a la 1790 se sigue exactamente la misma secuencia, con el octeto inferior del contador. La línea 1800 introduce el octeto inferior del registro HL, dentro del registro A, y en la línea 1810 le suma tres, para hacer que apunte al octeto superior del siguiente contador. Después, la línea 1820 actualiza L con el nuevo valor de A. En este momento hemos impreso sobre la pantalla el valor de un contador.

La línea 1830 carga B con un contador de bucle de cuatro, porque queremos mover el cursor a la siguiente parada de tabulación de la línea dos de la pantalla. Podríamos usado CURPOS de nuevo, pero aquí hemos usado un método diferente. El registro A se carga con nueve, que cuando se saca a la pantalla mueve el cursor una posición hacia la derecha. El registro B se prepara para ejecutar el bucle cuatro veces, porque hay ocho caracteres por cada zona de TAB, y hemos usado cuatro para imprimir los números, por lo tanto necesitamos avanzar cuatro posiciones para llegar al siguiente punto de parada de tabulador. Esto se hace en el bucle que comprende las líneas 1850 y 1860. Después de hacer esto, decrementamos el registro D en la línea 1870, para ver si se han impreso todos los contadores, si no, volvemos de nuevo a la línea 1680. Si se han completado, volvemos directamente a la rutina SWSCAN.

La figura 5.15 muestra la rutina TAPIO. Esta rutina escribe un fichero que contiene los valores actuales de los contadores, y el de

Fig 5.14 Acción de la instrucción RLD



Cuando se ejecuta la RLD, la dirección de memoria a la que apunta HL se desplaza a la derecha cuatro bits. Los cuatro bits inferiores se cargan con los cuatro bits inferiores del registro "A". Finalmente el contenido original de los cuatro bits superiores de la posición de memoria (HL) se colocan en los cuatro bits inferiores del registro "A".

PASS. El fichero en cinta se llama siempre "DATA LOG PASS xx" donde xx es el contador PASS en el momento en que se escribe el fichero. La operación de E/S la realiza una rutina de la ROM llamada CAS OUT DIRECT.

Para el CPC 664, debe cambiar el nombre de fichero de la línea 2370 a algo como "DLP" para construir un nombre legal de fichero. El resto permanecerá sin cambios. El programa comienza después de las explica-

ciones en los comentariosen, en la línea 1990, preparando una llamada a la rutina de la ROM que decide la velocidad a que se va a grabar la cinta. (Es equivalente al comando BASIC, SPEED WRITE). Esta rutina requiere que el valor de HL a la entrada contenga un valor equivalente a la mitad de la longitud de un bit a cero. También debe tener en el registro A, un valor que le diga cuanta compensación previa debe aplicar. En el caso de las líneas 1990 y 2000 los ponemos con los valores del comando BASIC, SPEED WRITE 1, ej 2K. En las líneas 2020 y 2030 nos aseguramos que el indicador de acarreo está limpio.

En las líneas 2370, 2380 y 2390 hay tres directrices DEFinir que representan un área que permite construir la cabecera completa del fichero para la rutina del cassette. Esta consta de NAMBUF, que forma la primera parte del nombre del fichero. Después PASCH y PASCL, que son los dos octetos donde se coloca la representación ASCII del valor del contador PASS.

```
1900
1910
1920  ; ** Esta es la sección TAPIO del programa de registro de datos
1930  ; ** Obtiene los valores de todos los contadores, y del contador
1940  ; ** PASS, y usa una de las rutinas de la ROM para escribirlos
1950  ; ** en fichero en cinta. Este fichero se llamará -
1960  ; ** "DATA LOG PASS nn" donde nn es el contador de pasadas
1970  ;
1980
1990  TAPIO:    LD   HL,167 ; Poner velocidad cassette rápida
2000            LD   A,50
2010            CALL #BC68  ; Llama a ROM, CAS SET SPEED
2020            LD   H,0          ; bit acarreo=0
2030            RL   H
2040            LD   HL,PASS      ; HL apunta a PASS
2050            LD   A,#30  ; Dígito superior = 3
2060            RLD
2070            LD   (PASCH),A    ; Dígito superior en nombre
2080            RLD
2090            LD   (PASCL),A    ; Dígito inferior en nombre
2100            RLD
2110            LD   HL,NAMBUF    ; HL dirección nombre fichero
2120            LD   B,16         ; Longitud nombre en B
2130            LD   DE,0   ; Buffer nulo
2140            CALL #BC8C  ; CAS OUT OPEN
2150            JP   C,OPENOK     ; OPEN OK
2160            LD   A,7          ; No, beep en A
2170            CALL #BB5A  ; para señalar OPEN fallido
2180            CALL #BC8F  ; Asegurarse que está cerrado
2190            JP   DLINIT ; Rearrancar
2200  OPENOK:   LD   HL,PASS      ; HL apunta a bloque control
2210            LD   DE,16  ; DE=tamaño datos escritura
2220            LD   BC,#FC00     ; Salvar valor en cabecera
2230            LD   A,2          ; tipo fichero= binario
2240            CALL #BC98  ; CAS OUT DIRECT
2250            CALL #BC8F  ; Cerrar fichero
2260            LD   IY,PASS      ; IY apunta a PASS
2270            LD   H,0
2280            RL   H            ; acarreo=0
2290            LD   A,(IY) ; A=contador PASS
2300            INC  A            ; Incrementar PASS
```

```
2310            DAA             ; Ajuste decimal
2320            LD   (IY),A ; devolverlo
2330            JP   RESTAR ; Rearrancar todo menso
2340  ; el contador PASS, saltando a RESTAR.
2350
2360
2370  NAMBUF:   DEFM "DATA LOG PASS "  ; Nombre fichero
2380  PASCH:    DEFB 0          ; Versión ASCII del
2390  PASCL:    DEFB 2          ; contador PASS
```

Fig 5.15

La línea 2040 introduce en HL el valor del contador PASS. Después, usando la misma técnica que usamos en DISPUP, usando la instrucción RLD (REF 129), convertimos el valor del contador PASS en dos octetos ASCII que se colocarán al final de NAMBUF, el nombre del fichero. En la línea 2110 se hace que HL apunte a NAMBUF, y después se pone B con dieciséis para decirle a la rutina de la ROM el número de caracteres que debe escribir en el nombre del fichero. Ponemos la dirección del 'buffer' a cero en la línea 2130. En la línea 2140 llamamos a la rutina de la ROM que maneja la apertura de ficheros, CAS_OUT_OPEN. Si se ha abierto correctamente el fichero (no estaba ya abierto o el usuario no ha pulsado ESC), la línea 2150 salta a la línea 2200. Si hubo algún problema con el OPEN, mandamos una señal acústica, e intentamos arreglar las cosas llamando a la rutina de CLOSE de ficheros de cassette, y volvemos a rearrancar el programa con un salto a DLINIT.

Ahora que hemos abierto el fichero, preparamos una llamada a la rutina CAS_OUT_DIRECT que escribe áreas de memoria en cinta. La línea 2200 carga la dirección del contador PASS en HL. (no olvide que el bloque de contadores comienza por debajo del contador PASS). Después ponemos en DE el valor de la longitud de los datos que vamos a escribir. La línea 2210 lo pone a 17 (PASS más los 16 octetos de los contadores). En la cabecera de un fichero de cinta en formato Amstrad, hay siempre 16 octetos que dicen dónde estaban en memoria los datos escritos en el fichero. Realmente no nos importan demasiado estos datos, así que se pone en la línea 2220 con la dirección de la pantalla, solo para divertirnos. La línea 2230 pone el tipo del fichero como binario, y después la línea 2240 manda toda esta información a la rutina de la ROM, y tiene lugar la escritura en la cinta. La línea 2250 llama a la rutina de CLOSE del fichero en cinta. Las líneas 2260 a 2320 hacen un incremento decimal en el contador PASS. Después la línea 2300 rearranca el programa haciendo un salto a RESTAR. Como ya dijimos anteriormente, las directrices de las líneas 2370 a 2390 solo reservan memoria para el área de la cabecera del fichero.

Con esto terminamos la descripción del 'software' para el dispositivo de registro de datos.

Los usuarios que deseen usar el 'software' como está, pero que no tienen el programa DEVPAC (debería tener un ensamblador - así que venda el gato!) deberán referirse a la siguiente sección de este capítulo. Finalmente, en la figura 5.16 le proporcionamos el listado del programa de registro de datos.

```
                    10
                    20 ;        ** Registro de Datos **
                    30
                    40 ;
                    50 ;
                    60 ;
                    70 ;   El programa está dividido en los siguientes
                    80 ;   módulos
                    90 ;
                   100
                   110 ;       1) "DLINIT" - Inicializa
                   120 ;       2) "SWSCAN" - Análiza los pulsadores
                   130 ;       3) "CPUDAT" - Actualiza los contadores
                   140 ;       4) "DISPUP" - Actualiza los contadores en pantalla
                   150 ;       5) "TAPIO"  - Sección de E/S a cinta
                   160
                   170 ; ** Además incorpora algunas
                   180 ; ** subrutinas más.
                   190
                   200
4E20               210         ORG  20000
4E20  00           220 PASS    DEFB 0          ; Contador de pasadas
4E21               230 CBASE   DEFS 16         ; 16 octetos para los contadores
4E31               240             DATLOG      ENT  $          ; Adelante!
                   250
4E31  01F7F8       260 DLINIT LD   BC,#F8F7
4E34  3EAE         270             LD    A,#AE     ; Cargar octeto de modo en PPI #2
4E36  ED79         280             OUT   (C),A
4E38  DD21204E     290             LD    IX,PASS   ; IX apunta al contador pasadas
4E3C  DD360000     300             LD    (IX),#0   ; Cero al contador de pasadas
4E40  1610         310 RESTAR LD   D,16  ; Preparar para poner a 0 los 8 contadores
4E42  DD21214E     320         LD   IX,CBASE   ; IX apunta a los contadores
4E46  DD360000     330 INIT00 LD   (IX),#0    ; Poner el contador a cero
4E4A  DD23         340         INC  IX         ; Incrementar IX
4E4C  15           350         DEC  D
4E4D  20F7         360         JR   NZ,INIT00  ; Repetir si hay más contadores
4E4F  3E04         370         LD   A,#04      ; Poner MODE 2 de pantalla
4E51  CD5ABB       380         CALL #BB5A      ; Usar la ROM para sacar el carácter
4E54  3E02         390         LD   A,#2       ; MODE 2, por favor
4E56  CD5ABB       400         CALL #BB5A
4E59  DD21914E     410         LD   IX,LABTXT  ; IX apunta a los textos de etiquetas
4E5D  1608         420         LD   D,8        ; D nos da 8 bucles
4E5F  0601         430         LD   B,#1       ; Usar B para número línea pantalla
4E61  0E01         440         LD   C,#1       ; Usar C para número columna pantalla
4E63  CD834E       450 INIT01 CALL CURPOS     ; Subrutina posición cursor
4E66  DD7E00       460 INIT02 LD   A,(IX)     ; Tomar un octeto de texto en A
4E69  DD23         470         INC  IX         ; Incrementar IX
4E6B  FEFF         480         CP   #FF        ; Ver si es fin de texto
4E6D  2805         490         JR   Z,INIT03   ; Si, saltar
4E6F  CD5ABB       500         CALL #BB5A      ; No, mandar el carácter
4E72  18F2         510         JR   INIT02     ; y repetir de nuevo
4E74  79           520 INIT03 LD   A,C         ; Cargar en A el contador de columnas
4E75  E607         530         AND  #07        ; Ver si estamos en punto de tabulador
4E77  2803         540         JR   Z,INIT04   ; Si, saltar
4E79  0C           550         INC  C          ; No, incrementar C
4E7A  18F8         560         JR   INIT03     ; Repetir unas cuantas veces
4E7C  0C           570 INIT04 INC  C          ; Mover C al TAB para la próxima vez
```

```
4E7D 15       580        DEC  D          ; Ver si se han sacado las 8 cabeceras
4E7E 20E3     590        JR   NZ,INIT01  ; No, saltar
4E80 C3C34E   600        JP   SWSCAN     ; Si, terminado. pasar a SWSCAN
              610
              620
4E83 3E1F     630 CURPOS LD   A,31  ; ** Subrutina de posicionar cursor
4E85 CD5ABB   640        CALL #BB5A ; C=Num col  B=línea pantalla
4E88 79       650        LD   A,C        ; Mandar columna
4E89 CD5ABB   660        CALL #BB5A
4E8C 78       670        LD   A,B        ; Mandar línea
4E8D CD5ABB   680        CALL #BB5A
4E90 C9       690        RET             ; Terminado, volver
              700
              710 ; ** Estos son solo ejemplos. Hasta 7 caracteres máximo
4E91 474F5252 720 LABTXT DEFM "GORRION"
4E98 FF       730        DEFB #FF
4E99 4C454348 740        DEFM "LECH"
4E9D FF       750        DEFB #FF
4E9E 41424144 760        DEFM "ABADE"
4EA3 FF       770        DEFB #FF
4EA4 43554552 780        DEFM "CUERV"
4EA9 FF       790        DEFB #FF
4EAA 48414C43 800        DEFM "HALC"
4EAE FF       810        DEFB #FF
4EAF 50494E5A 820        DEFM "PINZON"
4EB5 FF       830        DEFB #FF
4EB6 47415649 840        DEFM "GAVIOT"
4EBC FF       850        DEFB #FF
4EBD 4A494C47 860        DEFM "JILGU"
4EC2 FF       870        DEFB #FF
              880
              890
              900 ; **  SWSCAN - analiza los pulsadores para el registrador de
              910 ; **  datos. Esta rutina se activa después de la
inicilización
              920 ; **  y es la base del programa de registro de datos. Espera
              930 ; **  hasta que se oprima uno de los pulsadores conectados a
              940 ; **  la puerta B de PPI #2, después pasa el número del
              950 ; **  pulsador a la rutina de actualización de los
contadores.
              960 ; **  Si se pulsa la tecla correspondiente al carácter "#",
se
              970 ; **  escriben en cinta los valores actuales de los
              980 ; **  contadores y de PASS, por medio de la rutina TAPIO a la
              985 ; **  que salta esta rutina
              990
4EC3 CD09BB  1000 SWSCAN CALL #BB09      ; Llama a la ROM para ver si ha pulsa-
4EC6 3008    1010        JR   NC,SCAN01  ; do alguna tecla. Saltar si no
4EC8 FE23    1020        CP   35         ; ver si era #
4ECA CA494F  1030        JP   Z,TAPIO    ; Si, saltar
4ECD FEF8    1040        CP   #F8        ; ver si es CTRL/FLECHA ARRIBA
4ECF C8      1050        RET  Z          ; Si, devolver control
4ED0 01F5F8  1060 SCAN01 LD   BC,#F8F5   ; BC apunta a la puerta B de PPI #2
4ED3 ED58    1070        IN   E,(C)      ; Leer puerta B
4ED5 28EC    1080        JR   Z,SWSCAN   ; Si no se ha pulsado tecla, repetir
4ED7 1680    1090        LD   D,#80      ; Cargar una máscara de bits en D
4ED9 0608    1100        LD   B,8        ; Preparar ocho bucles
4EDB 7B      1110 SCAN03 LD   A,E        ; Poner en A el valor de la puerta B
4EDC A2      1120        AND  D          ; AND A con D
```

```
4EDD  2006      1130        JR   NZ,SCAN02  ; Saltar si el bit no es cero
4EDF  CB2A      1140        SRA  D          ; Desplazar D un bit a la derecha
4EE1  10F8      1150        DJNZ SCAN03     ; Decrem. B y saltar si es cero
4EE3  18DE      1160        JR   SWSCAN     ; Puerta B no 0 pero no pulsador, Hmm
4EE5  58        1170 SCAN02 LD   E,B        ; Salvar número pulsador en E
                1180 ; ** Nota el valor en E es el número del pulsador +1
                1190 ; ** porque se usa el contador de bucle como número de tecla
4EE6  01F5F8    1200        LD   BC,#F8F5   ; Esperamos a que se suelte la tecla
4EE9  ED78      1210 SCAN04 IN   A,(C)
4EEB  A2        1220        AND  D          ; Ver si todavía está pulsado
4EEC  20FB      1230        JR   NZ,SCAN04  ; No, bucle
4EEE  0680      1240        LD   B,#80 ; Retraso
4EF0  10FE      1250 SCAN05 DJNZ SCAN05     ; Bucle, 128 veces
4EF2  1D        1260        DEC  E          ; E=número real del contador
4EF3  C3F64E    1270        JP   CUPDAT     ; Ahora con E= número pulsador
                1280 ; pasamos a la rutina de actualización de contadores
                1290
                1300
                1310 ; ** Esta es la rutina de actualización de contadores-CUPDAT
                1320 ; ** A la entrada, el registro E contiene el número del
                1330 ; ** contador que hay que actualizar. El valor suministrado
                1340 ; ** se ajusta para convertirlo en un desplzamiento sobre el
                1350 ; ** bloque de contadores. El contador se incrementa en forma
                1360 ; ** decimal usando la instrucción DAA del Z80A
                1370 ;
4EF6  2600      1380 CUPDAT LD   H,0
4EF8  CB14      1390        RL   H          ; Limpiar el bit de acarreo
4EFA  21214E    1400        LD   HL,CBASE   ; HL apunta al bloque de contadores
4EFD  CB13      1410        RL   E          ; Multiplica E por 2
4EFF  7D        1420        LD   A,L        ; Copia L sobre A
4E00  83        1430        ADD  A,E        ; Añade el desplazamiento
4F01  6F        1440        LD   L,A        ; HL= dirección contador
4F02  7E        1450        LD   A,(HL)     ; Toma el valor del contador
4F03  3C        1460        INC  A          ; Lo incrementa
4F04  27        1470        DAA             ; Ajuste decimal
4F05  77        1480        LD   (HL),A     ; Lo vuelve a su lugar
4F06  3006      1490        JR   NC,ALLDUN  ; Ver si hay acarreo
4F08  3F        1500        CCF             ; Si, limpiarlo
4F09  23        1510        INC  HL         ; HL=siguiente posición contador
4F0A  7E        1520        LD   A,(HL)     ; Toma el valor del contador
4F0B  3C        1530        INC  A          ; Lo incrementa
4F0C  27        1540        DAA             ; Ajuste decimal
4F0D  77        1550        LD   (HL),A     ; Lo vuelve a su lugar
4F0E  00        1560 ALLDUN NOP             ; Ir a DISPUP
                1570
                1580
                1590 ; ** Esta rutina actualiza los contadores en pantalla -
                1600 ; ** DISPUP imprime los ocho contadores en la línea 2 de
                1610 ; ** la pantalla
                1620
4F0F  21224E    1630 DISPUP LD   HL,CBASE+1 ; HL=dir. octeto sup. prim. contador
4F12  0602      1640        LD   B,2,       ; Línea 2 de pantalla
4F14  0E01      1650        LD   C,1        ; Columna 1
4F16  CD834E    1660        CALL CURPOS     ; Posicionar el cursor
4F19  1608      1670        LD   D,8        ; Preparar 8 bucles
4F1B  3E30      1680 DISP01 LD   A,#30      ; Medio octeto sup. = 30
4F1D  ED6F      1690        RLD             ; Primer dígito en A
4F1F  CD5ABB    1700        CALL #BB5A      ; Mandarlo
4F22  ED6F      1710        RLD             ; Mover segundo dígito
```

```
4F24  CD5ABB    1720         CALL #BB5A      ; Mandarlo
4F27  ED6F      1730         RLD             ; Volver a estado original
4F29  2B        1740         DEC  HL         ; HL apunta a octeto inferior contador
4F2A  ED6F      1750         RLD             ; Mover el tercer dígito
4F2C  CD5ABB    1760         CALL #BB5A      ; mandarlo
4F2F  ED6F      1770         RLD             ; Mover cuarto dígito
4F31  CD5ABB    1780         CALL #BB5A      ; Mandarlo
4F34  ED6F      1790         RLD             ; Volver a estado original
4F36  7D        1800         LD   A,L        ; Incrementar L en tres
4F37  C603      1810         ADD  A,#3  ; apuntar al octeto superior contador sig.
4F39  6F        1820         LD   L,A        ; devolver valor a L
4F3A  0604      1830         LD   B,4        ; Cuatro bucles cursor a la derecha
4F3C  3E09      1840         LD   A,9        ; Bucle, cursor en sig. TAB
4F3E  CD5ABB    1850 DISP02  CALL #BB5A
4F41  10FB      1860         DJNZ DISP02     ; Bucle cuatro veces
4F43  15        1870         DEC  D          ; Terminados los contadores?
4F44  20D5      1880         JR   NZ,DISP01  ; No, repetir
4F46  C3C34E    1890         JP   SWSCAN     ; Volver a analizar pulsadores
                1900
                1910
                1920 ; ** Esta es la sección TAPIO del programa de registro de
                1930 ; ** datos. Obtiene los valores de todos los contadores, y
                1940 ; ** del contador PASS, y usa una de las rutina de la ROM
                1950 ; ** para escribirlos en fichero en cinta. Este fichero se
                1960 ; ** llamará -"DATA LOG PASS nn" donde nn es el contador de
                1970 ; ** pasadas
                1980
4F49  21A700    1990 TAPIO   LD   HL,167     ; Poner velocidad cassette rápida
4F4C  3E32      2000         LD   A,50
4F4E  CD68BC    2010         CALL #BC68      ; Llama a ROM, CAS SET SPEED
4F51  2600      2020         LD   H,0        ; bit acarreo=0
4F53  CB14      2030         RL   H
4F55  21204E    2040         LD   HL,PASS    ; HL apunta a PASS
4F58  3E30      2050         LD   A,#30      ; Dígito superior = 3
4F5A  ED6F      2060         RLD
4F5C  32B14F    2070         LD   (PASCH),A  ; Dígito superior en nombre
4F5F  ED6F      2080         RLD
4F61  32B24F    2090         LD   (PASCL),A  ; Dígito inferior en nombre
4F64  ED6F      2100         RLD
4F66  21A34F    2110         LD   HL,NAMBUF  ; HL dirección nombre fichero
4F69  0610      2120         LD   B,16       ; Longitud nombre en B
4F6B  110000    2130         LD   DE,0       ; Buffer nulo
4F6E  CD8CBC    2140         CALL #BC8C      ; CAS OUT OPEN
4F71  DA7F4F    2150         JP   C,OPENOK   ; OPEN OK
4F74  3E07      2160         LD   A,7        ; No, beep en A
4F76  CD5ABB    2170         CALL #BB5A      ; para señalar OPEN fallido
4F79  CD8FBC    2180         CALL #BC8F      ; Asegurarse que está cerrado
4F7C  C3314E    2190         JP   DLINIT     ; Rearrancar
4F7F  21204E    2200 OPENOK  LD   HL,PASS    ; HL apunta a bloque control
4F82  111000    2210         LD   DE,16      ; DE=tamaño datos escritura
4F85  0100FC    2220         LD   BC,#FC00   ; Salvar valor en cabecera
4F88  3E02      2230         LD   A,2        ; tipo fichero= binario
4F8A  CD98BC    2240         CALL #BC98 ; CAS OUT DIRECT
4F8D  CD8FBC    2250         CALL #BC8F ; Cerrar fichero
4F90  FD21204E  2260         LD   IY,PASS    ; IY apunta a PASS
4F94  2600      2270         LD   H,0
4F96  CB14      2280         RL   H          ; acarreo=0
4F98  FD7E00    2290         LD   A,(IY)     ; A=contador PASS
4F9B  3C        2300         INC  A          ; Incrementar PASS
```

```
4F9C  27        2310        DAA             ; Ajuste decimal
4F9D  FD7700    2320        LD   (IY),A     ; devolverlo
4FA0  C3404E    2330        JP   RESTAR     ; Rearrancar todo menso
                2340 ; el contador PASS, saltando a RESTAR.
                2350
                2360
4FA3  44415441  2370 NAMBUF DEFM "DATA LOG PASS "     ; Nombre fichero
4FB1  00        2380 PASCH  DEFB 0          ; Versión ASCII del
4FB2            2390 PASCL  DEFB 2          ; contador PASS

Pass 2 errors: 00

ALLDUN 4F0E  CBASE  4E21  CUPDAT 4EF6  CURPOS 4E83
DATLOG 4E31  DISP01 4F1B  DISP02 4F3E  DISPUP 4F0F
DLINIT 4E31  INIT00 4E46  INIT01 4E63  INIT02 4E66
INIT03 4E74  INIT04 4E7C  LABTXT 4E91  NAMBUF 4FA3
OPENOK 4F7F  PASCH  4FB1  PASCL  4FB2  PASS   4E20
RESTAR 4E40  SCAN01 4ED0  SCAN02 4EE5  SCAN03 4ED8
SCAN04 4EE9  SCAN05 4EF0  SWSCAN 4EC3  TAPIO  4F49


Table used:   371  from  1146
Executes: 20017
```

Fig 5.16

PERO, ¿AÚN NO TIENE ENSAMBLADOR?

Ya sabemos que hay muchos lectores que, una vez comprado su CPC, han agotado sus ahorros y no pueden hacer más compras destinadas a su ordenador. Esto les impide poder obtener algún programa extra, como un paquete ensamblador. En teoría por lo menos, es posible escribir un ensamblador en BASIC, pero es una tarea bastante compleja. Algo más práctico es buscar la forma de de usar los listados de código máquina publicados en revistas o libros, cargándolos desde el BASIC. Esto es más fácil, suponiendo que tenga dos tipos de información sobre el programa en código máquina. Uno es el listado del código máquina. Este se lista casi siempre en forma de vuelco hexadecimal o en listado de ensamblador. El segundo que debe conocer (Especialmente en el código de Z80 que no puede ser reubicable en memoria) es el lugar de la memoria en que se debe cargar el código máquina para que funcione correctamente. En los programas publicados siempre tendrá esta información.

El programa BASIC que se lista en la figura 5.17 se puede usar para tomar un área de memoria que contiene un programa en código máquina y convertido en un fichero compatible con BASIC, que tenga sentencias DATA, con el código máquina dentro de ellas. El programa escribirá el listado resultante a cinta o disco. Por supuesto, este programa solo le será útil si ya tiene el programa en código máquina disponible para cargarlo. Puede que este no sea su caso. Si está trabajando desde un simple listado, debe introducir cada código máquina dentro de sentencias DATA. (Recuerde que el formato "1000 DATA &4E,&80" es correcto - así que no tiene que realizar el tedioso trabajo de convertir cada código Hex a decimal). Cuando haya construido su lista de sentencias DATA, solo necesita poner unos cuantos READ y POKE al principio del programa y usar después una sentencia CALL, para ejecutar el

programa en código máquina que acaba de introducir en memoria. Esto se ilustra en la figura 5.18 que muestra cómo cargar le programa de registro de datos de forma correcta. No se olvide que tiene que informar al BASIC de que no debe usar la memoria donde ha cargado el código máquina, esto se hace con una sentencia MEMORY. A partir de ahora nos vamos a referir al programa BASIC, que contiene el programa en código máquina dentro de sentencias DATA como programa HEXBAS.

```
50    '**  Conversor de OBJETO a BASIC

100   '** Este programa toma un área de memoria que contiene un
       ** programa en código máquina, cargado previamente, y lo convierte
       ** en un fichero de texto compatible con el BASIC. Después puede
110   '** hacerse MERGE con otros ficheros BASIC.
120   CLS: MODE 2: LOCATE 4,5: INPUT "Nombre del fichero que va a crear en cinta";
      FIL$: SPEED WRITE 1: IF LEN(FIL$) > 16 THEN FIL$=LEFT$(FIL$,16):
      PRINT "Demasiado largo.. el nombre se ha truncado en ":FIL$
130   INPUT "Dirección de memoria donde empezar el vuelco ";AD
140   INPUT "Cuantos octetos hay que convertir? ";BYTES
150   INPUT "Número de la primer sentencia DATA ";STAT
160   DIM R(BYTES+15)
170   FOR I= AD TO (AD+BYTES): R(I- (AD-1))=PEEK(I): NEXT: OPENOUT FIL$
180   DUN=0
190   PRINT #9, STAT;" DATA";: FOR I=1 TO 10 : PRINT #9,R(I+DUN);:
      IF I= 10 THEN PRINT #9 ELSE PRINT #9,",";
200   NEXT I: STAT=STAT+10: DUN=DUN+10: IF DUN < BYTES THEN GOTO 190
      ELSE CLOSEOUT
32767 END
```

Fig 5.17 Programa de conversión de código máquina a sentencias DATA

```
500   '**  Programa de registro de datos, en BASIC ...  V1B Ene 1986
1000  D$="Esta es la versión HEXBAS del programa REGISTRO DE DATOS. Vea el ":
      E$="capítulo cinco para más detalles"

1100  MEMORY 19999: CLS: PRINT D$;E$;STRING$(5,10):
      PRINT "Cargando el código máquina en memoria ": FOR I=20000 TO 20404:
      READ R: POKE I,R: PRINT I;CHR$(13);: NEXT: CALL 20017
10000 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
10010 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 247 , 248
10020 DATA 62 , 174 , 237 , 121 , 221 , 33 , 32 , 78 , 221 , 54
10030 DATA 0 , 0 , 22 , 16 , 221 , 33 , 33 , 78 , 221 , 54
10040 DATA 0 , 0 , 221 , 35 , 21 , 32 , 247 , 62 , 4 , 205
10050 DATA 90 , 187 , 62 , 2 , 205 , 90 , 187 , 221 , 33 , 145
10060 DATA 78 , 22 , 8 , 6 , 1 , 14 , 1 , 205 , 131 , 78
10070 DATA 221 , 126 , 0 , 221 , 35 , 254 , 255 , 40 , 5 , 205
10080 DATA 90 , 187 , 24 , 242 , 121 , 230 , 7 , 40 , 3 , 12
10090 DATA 24 , 248 , 12 , 21 , 32 , 227 , 195 , 195 , 78 , 62
10100 DATA 31 , 205 , 90 , 187 , 121 , 205 , 90 , 187 , 120 , 205
10110 DATA 90 , 187 , 201 , 71 , 79 , 82 , 82 , 73 , 79 , 78
10120 DATA 255 , 76 , 69 , 67 , 72 , 255 , 65 , 66 , 65 , 68
10130 DATA 69 , 255 , 67 , 85 , 69 , 82 , 87 , 255 , 72 , 65
10140 DATA 76 , 67 , 255 , 80 , 73 , 78 , 90 , 79 , 78 , 255
10150 DATA 71 , 65 , 87 , 73 , 79 , 84 , 255 , 74 , 73 , 76
10160 DATA 71 , 85 , 255 , 205 , 9 , 187 , 48 , 8 , 254 , 35
10170 DATA 202 , 73 , 79 , 254 , 248 , 200 , 1 , 245 , 248 , 237
10180 DATA 88 , 40 , 236 , 22 , 128 , 6 , 8 , 123 , 162 , 32
```

```
10190 DATA 6 , 203 , 42 , 16 , 248 , 24 , 222 , 88 , 1 , 245
10200 DATA 248 , 237 , 120 , 162 , 32 , 251 , 6 , 128 , 16 , 254
10210 DATA 29 , 195 , 246 , 78 , 38 , 0 , 203 , 20 , 33 , 33
10220 DATA 78 , 203 , 19 , 125 , 131 , 111 , 126 , 60 , 39 , 119
10230 DATA 48 , 6 , 63 , 35 , 126 , 60 , 39 , 119 , 0 , 33
10240 DATA 34 , 78 , 6 , 2 , 14 , 1 , 205 , 131 , 78 , 22
10250 DATA 8 , 62 , 48 , 237 , 111 , 205 , 90 , 187 , 237 , 111
10260 DATA 205 , 90 , 187 , 237 , 111 , 43 , 237 , 111 , 205 , 90
10270 DATA 187 , 237 , 111 , 205 , 90 , 187 , 237 , 111 , 125 , 198
10280 DATA 3 , 111 , 6 , 4 , 62 , 9 , 205 , 90 , 187 , 16
10290 DATA 251 , 21 , 32 , 213 , 195 , 195 , 78 , 33 , 167 , 0
10300 DATA 62 , 50 , 205 , 104 , 188 , 38 , 0 , 203 , 20 , 33
10310 DATA 32 , 78 , 62 , 48 , 237 , 111 , 50 , 177 , 79 , 237
10320 DATA 111 , 50 , 178 , 79 , 237 , 111 , 33 , 163 , 79 , 6
10330 DATA 16 , 17 , 0 , 0 , 205 , 140 , 188 , 218 . 127 , 79
10340 DATA 62 , 7 , 205 , 90 , 187 , 205 , 143 , 188 , 195 , 49
10350 DATA 78 , 33 , 32 , 78 , 17 , 16 , 0 , 1 , 0 , 252
10360 DATA 62 , 2 , 205 , 152 , 188 , 205 , 143 , 188 , 253 , 33
10370 DATA 32 , 78 , 38 , 0 , 203 , 20 , 253 , 126 , 0 , 60
10380 DATA 39 , 253 , 119 , 0 , 195 , 64 , 78 , 68 , 65 , 84
10390 DATA 65 , 32 , 76 , 79 , 71 , 32 , 80 , 65 , 83 , 83
10400 DATA 32 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
```

Fig 5.18

UN PROGRAMA ÚTIL ESCRITO EN LENGUAJE ENSAMBLADOR

En el CPC 464 no hay forma (o no la he podido encontrar) de sacar un listado del catálogo de una cinta. El segundo programa que vamos a ver en lenguaje ensamblador (en menor detalle que el primero, ya que debe tener cierto dominio del ensamblador) es un programa que nos proporcionará esta facilidad. El programa tiene una primera parte opcional en BASIC, y se listan el ensamblaje y la versión HEXBAS.

Para comprender la operación del programa CATS PRINTER debe conocer primero la forma en que se almacenan los ficheros en cinta. En el libro SOFT 158 se da una información exhaustiva de este tema (ver sección 5.4). Cada bloque de datos en un fichero en cassette tiene una cabecera. Esta consta de 64 octetos que describen el fichero. Estos octetos se usan de la siguiente forma:

OCTETO 0-15 — Nombre del fichero

OCTETO 16 — Número del bloque

OCTETO 17 — Si no está a cero, el bloque que sigue a la cabecera es el último bloque del fichero.

OCTETO 18 — Octeto de tipo de fichero. Los bits de este octeto se asignan de la siguiente forma:

Bit 0 — Si es 1, el fichero está protegido

Bit 1-3 Tipo de fichero

000 = BASIC interno
001 = Binario
010 = Imagen de pantalla
011 = Fichero ASCII
1000-111 no asignados

OCTETO 19-20 Longitud del bloque de datos

OCTETO 21-22 Posición de los datos - Desde donde se escribió el fichero

OCTETO 23 Si no es cero, este es el primer bloque

OCTETO 24-25 Longitud del fichero en octetos

OCTETO 26-27 Dirección de ejecución del fichero, si es código máquina

OCTETO 28-63 Indefinidos, para el usuario.

Armados ahora con esta información veamos el programa CATS PRINTER. La versión en BASIC, con la sección HEXBAS incluida, se lista en la figura 5.19. La sección BASIC le pregunta por la fecha y el nombre del cassette, y la cara que va a catalogar. Estos datos se imprimen como una cabecera sobre el listado de los ficheros.

```
1000 '                    ** CATS PRINTER **
      Este programa imprime un catálogo de los ficheros de una cinta.
      Esta es la parte previa a un pequeño programa en código máquina que
1100 ' se introduce en memoria a partir de la dirección 25000. Tenga en
      cuenta los siguientes puntos; a diferencia del comando CAT del BASIC,
      no lee los datos de los ficheros, solo lee las cabeceras. El programa no
1200 ' puede cohabitar con otro programa BASIC, si lo intenta pueden ocurrir
      resultados imprevisibles. La memoria intermedia de cassette es definida
      por la sección de código máquina, para que comience en 40000.
1210 ON BREAK GOSUB 1600
1300 CLS: RESTORE: MEMORY 24999: FOR I=25000 TO 25315: READ P:
     POKE I,P: NEXT I: INPUT "Teclee la fecha de hoy ";DOY$:
     ' introduce en memoria la sección de código máquina. Pone la fecha
1305 INPUT "Teclee el nombre de esta cinta ";D$:
     IF D$="" THEN D$="Cinta desconocida"
1400 INPUT "De qué cara vamos a listar los ficheros? ";S$:
     IF S$="" THEN S$="No especificada"
1500 PRINT #8: PRINT #8,,"Catalogo de la cinta - ";D$;
     ": CARA ";S$;" ";DOY$: PRINT #8: CALL 25000: PRINT #8
     'Imprime el nombre del fichero, número de cara y fecha como cabecera
1600 PRINT "Catalogo completado": PRINT #8,STRING$(25,45):
     'Rearrancar cuando el usuario pulse ESC, después de catalogar
      la cinta.
1610 PRINT : INPUT "Quiere catalogar más cintas? ";R$:
     IF UPPER$(LEFT$(R$,1))="S" THEN GOTO 1305 ELSE GOTO 32767
10000 DATA 33 , 64 , 156 , 17 , 64 , 0 , 62 , 44 , 63 , 205
10010 DATA 161 , 188 , 56 , 8 , 205 , 153 , 98 , 254 , 0 , 32
10020 DATA 235 , 201 , 17 , 64 , 156 , 38 , 16 , 26 , 254 , 0
10030 DATA 32 , 2 , 62 , 32 , 205 , 207 , 98 , 19 , 37 , 32
```

```
10040 DATA 242 , 62 , 5 , 205 , 153 , 98 , 26 , 39 , 103 , 203
10050 DATA 63 , 203 , 63 , 203 , 63 , 203 , 63 , 246 , 48 , 205
10060 DATA 207 , 98 , 124 , 230 , 15 , 246 , 48 , 205 , 207 , 98
10070 DATA 19 , 19 , 62 , 48 , 205 , 153 , 98 , 26 , 230 , 7
10080 DATA 246 , 48 , 205 , 207 , 98 , 62 , 64 , 205 , 153 , 98
10090 DATA 24 , 164 , 0 , 13 , 10 , 10 , 67 , 97 , 116 , 97
10100 DATA 108 , 111 , 103 , 111 , 32 , 97 , 98 , 111 , 114 , 116
10110 DATA 97 , 100 , 111 , 32 , 112 , 111 , 114 , 32 , 42 , 32
10120 DATA 66 , 82 , 69 , 65 , 75 , 32 , 42 , 32 , 255 , 1
10130 DATA 0 , 42 , 42 , 32 , 69 , 114 , 114 , 111 , 114 , 32
10140 DATA 99 , 105 , 110 , 116 , 97 , 45 , 98 , 105 , 116 , 32
10150 DATA 109 , 117 , 121 , 32 , 108 , 97 , 114 , 103 , 111 , 32
10160 DATA 42 , 42 , 255 , 2 , 42 , 42 , 32 , 69 , 114 , 114
10170 DATA 111 , 114 , 32 , 99 , 105 , 110 , 116 , 97 , 45 , 32
10180 DATA 101 , 114 , 114 , 111 , 114 , 32 , 99 , 104 , 101 , 99
10190 DATA 107 , 115 , 117 , 109 , 32 , 42 , 42 , 255 , 5 , 32
10200 DATA 78 , 117 , 109 , 46 , 32 , 98 , 108 , 111 , 113 , 117
10210 DATA 101 , 32 , 32 , 255 , 64 , 32 , 67 , 111 , 114 , 114
10220 DATA 101 , 99 , 116 , 111 , 10 , 13 , 255 , 48 , 32 , 84
10230 DATA 105 , 112 , 111 , 32 , 102 , 105 , 99 , 104 , 61 , 255
10240 DATA 0 , 253 , 229 , 213 , 245 , 253 , 33 , 4 , 98 , 253
10250 DATA 86 , 0 , 253 , 35 , 186 , 40 , 19 , 253 , 35 , 253
10260 DATA 203 , 0 , 126 , 40 , 248 , 253 , 35 , 253 , 86 , 0
10270 DATA 254 , 0 , 40 , 227 , 24 , 229 , 253 , 126 , 0 , 205
10280 DATA 207 , 98 , 253 , 35 , 253 , 203 , 0 , 126 , 40 , 242
10290 DATA 241 , 209 , 253 , 225 , 201 , 197 , 6 , 245 , 237 , 72
10300 DATA 203 , 113 , 32 , 250 , 6 , 239 , 203 , 255 , 237 , 121
10310 DATA 203 , 191 , 237 , 121 , 193 , 201 , 0 , 0 , 0
32767 END
```

Fig 5.19

El funcionamiento de la sección BASIC es autoexplicativa. El código máquina que introduce en memoria en la dirección 25000 es el resultado del ensamblaje del programa listado en la figura 5.20.

```
 10 ;       CAT PRINTER
 20 ;  imprime el catálogo de una cinta.
 40 ; Este programa llama a la rutina CAS_READ de la ROM,
 50 ; después de cada llamada, analiza la información de la cabecera
 60 ; cargada desde la cinta, e imprime en forma de catálogo la
 70 ; información que contiene.
 80           ORG  25000
 90           ENT  $
100 CATPO:    LD   HL,40000    ; Mem. int. en 40000 (d)
110           LD   DE,64  ; lee 64 octetos
120           LD   A,#2C  ; carácter sincronismo cabecera
130           CCF              ; limpia acarreo
140           CALL #BCA1  ; CAS_READ, acarreo=1 si ok
150           JR   C,READOK    ; OK, salta a READOK
160           CALL MESOUT ; NOK, manda mensaje
170           CP   #0          ; número mensaje=0?
180           JR   NZ,CATPO    ; No, pruebe de nuevo
190           RET              ; volver al BASIC
200 READOK:   LD   DE,40000    ; Si ha leido bien, debemos
210 ;            imprimir el nombre del fichero
220 ;              DE apunta a la cabecera
```

```
230             LD   H,16   ; 16 caracteres en nombre fichero
240  CATP4:     LD   A,(DE) ; toma un octeto
250             CP   #0          ; ver si es nulo
260             JR   NZ,CATP6    ; no, saltar
270             LD   A,#20  ; rellenar con espacios
280  CATP6:     CALL PRINT  ; imprimir el carácter
290             INC  DE          ; Incrementar apuntador
300             DEC  H           ; decrementar H - contador bucle
310             JR   NZ,CATP4
320  ; Una vez impreso el nombre del fichero debemos imprimir
330  ; el número de bloque que está apuntado por IY.
340  SHOBLK:    LD   A,05   ; A=código del mensaje "Núm. bloque"
350             CALL MESOUT ; imprimirlo
360             LD   A,(DE) ; Tomar número de bloque
370             DAA              ; ponerlo en BCD
380             LD   H,A         ; Copiar el resultado a H
390             SRL  A           ; dígito alto a dígito bajo
400             SRL  A
410             SRL  A
420             SRL  A           ; Hecho!
430             OR   #30         ; convertirlo en ASCII
440             CALL PRINT  ; imprimirlo
450             LD   A,H         ; recuperar num bloque
460             AND  #0F         ; extraer dígito inferior
470             OR   #30         ; convertirlo en ASCII
480             CALL PRINT  ; imprimirlo
490             INC  DE          ; DE apuntará a tipo fichero
500             INC  DE          ; que está 2 octetos adelante
510  ;               Ahora se debe indicar el tipo de fichero
520  ;               Se hace cambiando los cuatro bits superiores
530  ;               del octeto de tipo a 0011, esto lo convierte
540  ;               en un número ASCII, listo para MESOUT.
550  SHOTYP:    LD   A,#30  ; Carga num. mens. de "Tipo fich="
560             CALL MESOUT ; lo imprime
570             LD   A,(DE) ; toma octeto tipo fichero
580             AND  #07         ; cero los 5 bits superiores
590             OR   #30         ; convierte los 4 inferiores en num. mens.
600             CALL PRINT  ; imprime tipo fichero en números
610  HOORAY:    LD   A,#40  ; toma num. mens. de "Correct"
620             CALL MESOUT ; lo imprime
630             JR   CATP0  ; y empieza de nuevo
640  ;
650  ;
660  ;
670  ;      MESTAB contiene los mensajes
680  ; que imprime la subrutina MESOUT. el formato es: num. mens.;
690  ; texto; Hex FF; MESNUM: hasta que MESNUM=0 para terminar
700  MESTAB:    DEFB #00,#0D,#0A,#0A
710             DEFM "Catálogo abortado por * BREAK * "
720             DEFB #FF,#01,#00
730             DEFM "** Error cinta-bit muy largo **"
740             DEFB #FF,#02
750             DEFM "** Error cinta- error checksum **"
760             DEFB #FF,#05
770             DEFM " Num. bloque  "
780             DEFB #FF,#40
790             DEFM " Correcto"
800             DEFB #0A,#0D
810             DEFB #FF,#30
```

```
820           DEFM " Tipo Fich="
830           DEFB #FF,#00      ; fin de la tabla de mensajes
840 ;
850 ;
860 ;         MESOUT: imprime los mensajes de error cuyo
870 ;              código encuentra en el registro A.
880 ;              La tabla MESTAB debe ser añadida antes de usarla
890 ;              La etiqueta OUTDEV: debe ser para un controlador
900 ;              de dispositivo de salida, ej impresora, pantalla,  E/S
910 ; se mira en la tabla de mensajes MESTAB
920 ; que tiene el formato; Num. Mens,Texto,#FF. Un octeto nulo
930 ; termina la tabla de mensajes.
940 ; A la entrada, A=número de mensaje a imprimir
950 ; se restauran todos los registros al salir
960 MESOUT:   PUSH IY           ; salvar IY
970           PUSH DE           ; salvar DE
980           PUSH AF           ; salvar AF
990 MES001:   LD   IY,MESTAB    ; IY apunta a MESTAB
1000 MES002:  LD   D,(IY) ; carga en D num. mens.
1010          INC  IY           ; incrementa IY
1020          CP   D            ; ver si es nuestro mensaje
1030          JR   Z,OUTMES     ; si, saltar
1040 MES003:  INC  IY           ; incremetar IY sobre la tabla
1050          BIT  7,(IY) ; bit 7 puesto?
1060          JR   Z,MES003     ; no, saltar
1070          INC  IY           ; ver el siguiente octeto
1080          LD   D,(IY)
1090          CP   #0           ; ver si es nulo
1100          JR   Z,MES001     ; so, falló la búsqueda
1110 ; mandar mensaje BREAK
1120          JR   MES002 ; no, ver si num. mens. ok
1130 OUTMES:  LD   A,(IY) ; toma un carácter
1140          CALL PRINT  ; lo imprime
1150          INC  IY           ; incrementa apuntador tabla
1160          BIT  7,(IY) ; terminado?
1170          JR   Z,OUTMES     ; no, seguir
1180          POP  AF
1190          POP  DE
1200          POP  IY
1210 GOBACK:  RET               ; limpiar pila y retornar
1220 ;
1230 ;
1240 ;              PRINT
1250 ;      imprime los caracteres del registro A
1260 ;   NOTA * LA RUTINA SE CUELGA SI LA IMPRESORA NO ESTA PREPARADA
1270 PRINT:   PUSH BC           ; Salvar BC
1280          LD   B,#F5  ; impresora preparada?, bit -8255 puerta B
1290 WAIT:    IN   C,(C)  ; obtiene estado
1300          BIT  6,C
1310          JR   NZ,WAIT      ; bit 6=1, impresora no preparada, bucle
1320          LD   B,#EF  ; preparada, BC apunta a datos
1330          SET  7,A          ; Poner bit sincronismo
1340          OUT  (C),A
1350          RES  7,A          ; mandarlo, y restaurar bit
1360          OUT  (C),A  ; mandar sincronismo=0
1370          POP  BC           ; restaurar BC
1380          RET               ; y adios
```

Fig 5.20

La parte del programa generada por el ensamblador lee las cabeceras del cassette, y después imprime un resumen de información de una línea por cada cabecera de fichero encontrada. La figura 5.21 muestra el formato del informe producido por el programa.

```
          Catalogo de la cinta - DEMO: CARA 1  2 Agosto 1986

CAT PRINTER.EXE  Num. bloque  01 Tipo fich=6 Correcto
CAT PRINTER.EXE  Num. bloque  02 Tipo fich=6 Correcto
SPEECH DEMO AUTO Num. bloque  01 Tipo fich=6 Correcto
SPEECH DEMO AUTO Num. bloque  02 Tipo fich=6 Correcto
KEY MAT DEMO     Num. bloque  01 Tipo fich=1 Correcto
KEY MAT DEMO     Num. bloque  02 Tipo fich=1 Correcto
SERIAL XMIT DEMO Num. bloque  01 Tipo fich=1 Correcto
SERIAL XMIT DEMO Num. bloque  02 Tipo fich=1 Correcto
MEMORY DUMPER    Num. bloque  01 Tipo fich=6 Correcto
MINEFIELD !      Num. bloque  01 Tipo fich=6 Correcto
MINEFIELD !      Num. bloque  02 Tipo fich=6 Correcto
MINEFIELD !      Num. bloque  03 Tipo fich=6 Correcto
MINEFIELD !      Num. bloque  04 Tipo fich=6 Correcto

Catálogo abortado por * BREAK *----------------------------
```

Fig 5.21 Informe del programa CATS PRINTER

Usted puede volver a dirigir fácilmente la salida del programa a la pantalla, o a una puerta serie, ó a cualquier otro dispositivo (excepto cassette) sustituyendo su propia versión de la subrutina PRINT. Por ejemplo, si quiere la salida del programa en la pantalla del CPC, la subrutina PRINT debe cambiarse para que sea así:

```
1270  PRINT:    CALL #BB5A  ; inprimir el carácter en A
1280            RET           ; y retornar
```

y puede acomodarla a sus propios requerimientos, La rutina PRINT que hemos puesto, hace más complicada la salida de caracteres a la impresora, pero querido lector, es por su beneficio. La idea es mostrar cómo se maneja directamente la puerta de la impresora para que conozca cómo funciona. La rutina se puede reemplazar directamente por:

```
1270  PRINT:    CALL #BD31  ; Rutina de la ROM que imprime el carácter en A
1280            RET           ; y retornar
```

Este cambio no se ha hecho en el programa listado. Como ya se dijo en un capitulo anterior, la dirección de E/S Hex F5xx (donde xx puede ser cualquier valor) es la dirección de la puerta de impresora. El bit siete debe ponerse a cero mientras que el código del carácter a imprimir aparece en los bits 0-6.

Vayamos ahora a la sección de ensamblador del programa CATS PRINTER, rutina a rutina. Al principio CATPO especifica la dirección de la memoria intermedia del cassette, y el carácter especial de sincronismo (esto es, el que espera la rutina de la ROM como primer carácter de la cabecera). A continuación llama a la rutina de la dirección Hex BCA1, que lee la cabecera en la memoria intermedia en la dirección decimal 40000. Cuando vuelve de esta rutina, si se ha cargado la cabecera sin errores el bit de acarreo tendrá un uno . Si ha ocu-

rrido algún error, llama a la rutina MESOUT. Si el error fue que el usuario pulsó la tecla ESC, volverá al BASIC. Si todo va bien se usará el registro DE como apuntador a la cabecera cargada, y lo primero que se extrae e imprime es el nombre del fichero. Los octetos no usados del nombre del fichero se almacenan como blancos. Para conseguir el alineamiento, todos los caracteres nulos se almacenan como blancos. A continuación se convierte el número de bloque a ASCII y se imprime.

Lo siguiente es extraer el tipo de fichero de la cabecera cargada. Se imprime como un número con el siguiente significado:

0 = BASIC interno
2 = Binario
4 = Imagen de pantalla
6 = fichero en formato ASCII

Finalmente se imprime la palabra "Correcto" para indicar que la cabecera se ha cargado sin errores. Es importante conocer las limitaciones de este programa. Solo lee las cabeceras, no se verifican los bloques de datos que describen. Esto se debe a que el programa ha sido concebido como una forma de obtener un informe del contenido de la cinta, no como un programa de verificación, esto último lo hace el comando CAT.

Las dos subrutinas MESOUT y PRINT están explicadas en los comentarios del listado. El formato de la tabla de mensajes, MESTAB, que contiene todos los mensajes que necesita lanzar el programa, tiene la siguiente estructura:

Número mensaje | Texto | #FF | Número mensaje | Texto | #FF etc.

La tabla termina cuando el octeto siguiente a un #FF, es cero. MESOUT recorre MESTAB hasta que encuentra el número de mensaje que coincide con el que se ha mandado en el registro A. La rutina PRINT ya se ha explicado, y como ya hemos dicho es fácil de reemplazar con una CALL a otro rutina.

CONFESIÓN DEL AUTOR

Para corroborar mi anterior disertación sobre la necesidad de hacer un diagrama de flujo antes de escribir el programa en ensamblador, le voy a confesar algo. El programa CATS PRINTER funciona. Sin embargo no es un programa elegante, y la razón es - como habrá adivinado - que la escribí directamente en el teclado. Esto lo hice con la creencia de que así podría terminarla más rápidamente. Sin embargo tarde el doble. Una vez que la longitud del programa pasa de cierto punto, debe imprimirlo, hacer las correcciones e introducirlas. De ahora en adelante prometo practicar lo que predico, seguir la regla de los diagramas de flujo. La figura 5.22 es el listado ensamblado del programa CATS PRINTER.

```
10 ;       CAT PRINTER
20 ;  imprime el catálogo de una cinta.
40 ; Este programa llama a la rutina CAS_READ de la ROM,
50 ; después de cada llamada, analiza la información de la
```

```
                60 ; cabecera cargada desde la cinta, e imprime en forma de
                70 ; catálogo la información que contiene.
61A8            80        ORG  25000
61A8            90        ENT  $
61A8  21409C   100 CATPO   LD   HL,40000   ; Mem. int. en 40000 (d)
61AB  114000   110         LD   DE,64 ; lee 64 octetos
61AE  3E2C     120         LD   A,#2C ; carácter sincronismo cabecera
61B0  3F       130         CCF              ; limpia acarreo
61B1  CDA1BC   140         CALL #BCA1 ; CAS_READ, acarreo=1 si ok
61B4  3808     150         JR   C,READOK    ; OK, salta a READOK
61B6  CD9962   160         CALL MESOUT      ; NOK, manda mensaje
61B9  FE00     170         CP   #0          ; número mensaje=0?
61BB  20EB     180         JR   NZ,CATPO    ; No, pruebe de nuevo
61BD  C9       190         RET              ; volver al BASIC
61BE  11409C   200 READOK  LD   DE,40000    ; Si ha leido bien, debemos
                210 ;               imprimir el nombre del fichero
                220 ;              DE apunta a la cabecera
61C1  2610     230         LD   H,16  ; 16 caracteres en nombre fichero
61C3  1A       240 CATP4   LD   A,(DE)      ; toma un octeto
61C4  FE00     250         CP   #0          ; ver si es nulo
61C6  2002     260         JR   NZ,CATP6    ; no, saltar
61C8  3E20     270         LD   A,#20 ; rellenar con espacios
61CA  CDCF62   280 CATP6   CALL PRINT : imprimir el carácter
61CD  13       290         INC  DE          ; Incrementar apuntador
61CE  25       300         DEC  H           ; decrementar H - contador bucle
61CF  20F2     310         JR   NZ,CATP4
                320 ; Una vez impreso el nombre del fichero debemos imprimir
                330 ; el número de bloque que está apuntado por IY.
61D1  3E05     340 SHOBLK LD   A,05  ; A=código del mensaje "Núm. bloque"
61D3  CD9962   350         CALL MESOUT      ; imprimirlo
61D6  1A       360         LD   A,(DE)      ; Tomar número de bloque
61D7  27       370         DAA              ; ponerlo en BCD
61D8  67       380         LD   H,A         ; Copiar el resultado a H
61D9  CB3F     390         SRL  A           ; dígito alto a dígito bajo
61DB  CB3F     400         SRL  A
61DD  CB3F     410         SRL  A
61DF  CB3F     420         SRL  A           ; Hecho!
61E1  F630     430         OR   #30         ; convertirlo en ASCII
61E3  CDCF62   440         CALL PRINT ; imprimirlo
61E6  7C       450         LD   A,H         ; recuperar num bloque
61E7  E60F     460         AND  #0F         ; extraer dígito inferior
61E9  F630     470         OR   #30         ; convertirlo en ASCII
61EB  CDCF62   480         CALL PRINT ; imprimirlo
61EE  13       490         INC  DE          ; DE apuntará a tipo fichero
61EF  13       500         INC  DE          ; que está 2 octetos adelante
                510 ;              Ahora se debe indicar el tipo de fichero
                520 ;              Se hace cambiando los cuatro bits superiores
                530 ;              del octeto de tipo a 0011, esto lo convierte
                540 ;              en un número ASCII, listo para MESOUT.
61F0  3E30     550 SHOTYP LD   A,#30 ; Carga num. mens. de "Tipo fich="
61F2  CD9962   560         CALL MESOUT      ; lo imprime
61F5  1A       570         LD   A,(DE)      ; toma octeto tipo fichero
61F6  E607     580         AND  #07         ; cero los 5 bits superiores
61F8  F630     590         OR   #30   ; convierte los 4 inferiores en num. mens.
61FA  CDCF62   600         CALL PRINT ; imprime tipo fichero en números
61FD  3E40     610 HOORAY LD   A,#40 ; toma num. mens. de "Correcto"
61FF  CD9962   620         CALL MESOUT      ; lo imprime
6202  18A4     630         JR   CATPO ; y empieza de nuevo
                640 ;
```

```
                650 ;
                660 ;       MESTAB contiene los mensajes
                670 ; que imprime la subrutina MESOUT. el formato es num. mens.;
                680 ; texto; Hex FF; MESNUM hasta que MESNUM=0 para terminar
                690 ;
6204  000D0A0A  700 MESTAB DEFB #00,#0D,#0A,#0A
6208  43617461  710       DEFM "Catalogo abortado por * BREAK * "
6228  FF0100    720       DEFB #FF,#01,#00
622B  2A2A2045  730       DEFM "** Error cinta-bit muy largo **"
624A  FF02      740       DEFB #FF,#02
624C  2A2A2045  750       DEFM "** Error cinta- error checksum **"
626D  FF05      760       DEFB #FF,#05
626F  204E756D  770       DEFM " Num. bloque  "
627D  FF40      780       DEFB #FF,#40
627F  20436F72  790       DEFM " Correcto"
6288  0A0D      800       DEFB #0A,#0D
628A  FF30      810       DEFB #FF,#30
628C  20546970  820       DEFM " Tipo Fich="
6297  FF00      830       DEFB #FF,#00    ; fin de la tabla de mensajes
                840 ;
                850 ;
                860 ; MESOUT imprime los mensajes de error cuyo
                870 ; código encuentra en el registro A.
                880 ; La tabla MESTAB debe ser añadida antes de usarla
                890 ; La etiqueta OUTDEV debe ser para un controlador
                900 ; de dispositivo de salida, ej impresora, pantalla, E/S
                910 ; se mira en la tabla de mensajes MESTAB
                920 ; que tiene el formato; Num. Mens,Texto,#FF. Un octeto nulo
                930 ; termina la tabla de mensajes.
                940 ; A la entrada, A=número de mensaje a imprimir
                950 ; se restauran todos los registros al salir
6299  FDE5      960 MESOUT PUSH IY          ; salvar IY
629B  D5        970       PUSH DE           ; salvar DE
629C  F5        980       PUSH AF           ; salvar AF
629D  FD210462  990 MES001 LD   IY,MESTAB   ; IY apunta a MESTAB
62A1  FD5E00   1000 MES002 LD   D,(IY)      ; carga en D num. mens.
62A4  FD23     1010       INC  IY           ; incrementa IY
62A6  BA       1020       CP   D            ; ver si es nuestro mensaje
62A7  2813     1030       JR   Z,OUTMES     ; si, saltar
62A9  FD23     1040 MES003 INC  IY          ; incremetar IY sobre la tabla
62AB  FDCB007E 1050       BIT  7,(IY)       ; bit 7 puesto?
62AF  29F8     1060       JR   Z,MES003     ; no, saltar
62B1  FD23     1070       INC  IY           ; ver el siguiente octeto
62B3  FD5E00   1080       LD   D,(IY)
62B6  FE00     1090       CP   #0           ; ver si es nulo
62B8  28E3     1100       JR   Z,MES001     ; so, falló l abúsqueda
               1110 ; mandar mensaje BREAK
62BA  18E5     1120       JR   MES002       ; no, ver si num. mens. ok
62BC  FD7E00   1130 OUTMES LD   A,(IY)       ; toma un carácter
62BF  CDCF62   1140       CALL PRINT ; lo imprime
62C2  FD23     1150       INC  IY           ; incrementa apuntador tabla
62C4  FDCB007E 1160       BIT  7,(IY)       ; terminado?
62C8  28F2     1170       JR   Z,OUTMES     ; no, seguir
62CA  F1       1180       POP  AF
62CB  D1       1190       POP  DE
62CC  FDE1     1200       POP  IY
62CE  C9       1210 GOBACK RET              ; limpiar pila y retornar
               1220 ;
               1230 ;
```

```
                1240 ;                  PRINT
                1250 ;       imprime los caracteres del registro A
                1260 ; NOTA: LA RUTINA SE CUELGA SI LA IMPRESORA NO ESTA PREPARADA
62CF  C5        1270 PRINT  PUSH BC         ; Salvar BC
62D0  06F5      1280        LD   B,#F5 ; impresora preparada?, bit -8255 puerta B
62D2  ED48      1290 WAIT   IN   C,(C) : obtiene estado
62D4  CB71      1300        BIT  6,C
62D6  20FA      1310        JR   NZ,WAIT    ; bit 6=1,impresora no prep., bucle
62D8  06EF      1320        LD   B,#EF ; preparada, BC apunta a datos
62DA  CBFF      1330        SET  7,A        ; Poner bit sincronismo
62DC  ED79      1340        OUT  (C),A
62DE  CB8F      1350        RES  7,A        ; mandarlo, y restaurar bit
62E0  ED79      1360        OUT  (C),A ; mandar sincronismo=0
62E2  C1        1370        POP  BC         : restaurar BC
62E3  C9        1380        RET             ; y adios

Pass 2 errors: 00

CATP0  61A8  CATP4  61C3  CATP6  61CA  GOBACK 61CE
HOORAY 61FD  MES001 619D  MES002 61A1  MES003 61A9
MESOUT 6199  MESTAB 6104  OUTMES 61BC  PRINT  62CF
READOK 61BE  SHOBLK 61D1  SHOTYP 61F0  WAIT   62D2


Table Used:   215  from   685
Executes: 25000
```

Fig 5.22

INSTRUCCIONES QUE NO DEBE USAR

Hay ciertos tipos de instrucciones en el juego del Z80A que no se deben usar en las máquinas CPC. Son, principalmente, las que manejan las interrupciones y el juego alternado de registros.

Como ya dijimos en el capítulo cuatro, la interrupción que va al Z80A se usa como señal para ejecutar cierto número de acciones, incluyendo el análisis del teclado. Si se desactivan las interrupciones con un programa en código máquina, mediante la instrucción DI (REF 30), y no se reactivan antes de que termine el programa, no se podrá teclear nada. Por lo tanto, la recomendación es que, a menos que no le quede otra alternativa, evite usar la instrucción DI.

Otro grupo de instrucciones que se deben evitar en el CPC son las que afectan al juego alternado de registros. Un ejemplo es la instrucción EX AF,AF' (REF 33). La ROM del CPC usa el juego alternado de registros para mantener varios contadores de las rutinas de interrupción, así como apuntadores al espacio de E/S, etc. Si usa usted el juego alternativo de registros puede hacer que falle el ordenador. (Tampoco se debe usar nunca EXX - REF 38).

Los comentarios de los dos párrafos anteriores no serán aplicables si usted desarrolla el 'software' para ejecutar una aplicación independiente de la ROM. En ese caso, deberá construir una configuración 'hardware' aparte, en la que pondrá las EPROM que haya grabado, con el grabador de EPROM del capítulo tres. A medida que se de cuenta de que la sección del programa que contenga estas instrucciones no se puede ejecutar en el CPC, puede que decida usarlas en programas desa-

rrollados para su uso fuera de la máquina CPC.

Otro punto sobre el uso del CPC como herramienta de prueba para desarrollo de programas, es que debe tener extremo cuidado al asignar las direcciones de E/S para que, en el proyecto, las E/S sean compatibles con los huecos en el uso de direcciones de E/S en el CPC. Si no lo tiene, no podrá usar el CPC como herramienta de depuración para sus programas. Teniendo como referencia la tabla de direcciones de E/S que viene en el libro SOFT 158, podrá comprender como trabaja el direccionamiento de E/S, y le será posible encontrar un hueco para que su propio 'hardware' use rangos de direcciones libres.

EL PROGRAMA DE DEPURACIÓN MONA3

Cuando compró el paquete DEVPAC, realmente adquirió dos programas. El ensamblador GENA3 y el programa de depuración MONA3. MONA3 es sumamente útil, porque, entre otras cosas, le permite ejecutar un programa paso a paso, con una pantalla que le muestra el contenido de todos los registros de la CPU después de cada instrucción. La única limitación para la ejecución paso a paso es que, si usa en su programa instrucciones CALL para llamar a subrutinas de la ROM, puede encontrarse que las subrutinas de la ROM contienen instrucciones como EXX (ver la sección anterior). Este pequeño problema se puede solventar con la inserción de puntos de ruptura. Un punto de ruptura se inserta usando el MONA3, y le permite ejecutar su programa hasta un punto concreto, después del cual retorna a la pantalla de registros del MONA3. Esto le da mayor potencia para ejecutar un programa hasta un punto en que tiene un problema y ver el contenido de los registros en el punto del problema.

MONA3 tiene también una facilidad de desensamblador. Esto significa que puede cargar cualquier programa en código máquina en la memoria, y decirle al MONA3 dónde empieza, para que produzca un listado del código máquina del tipo del lenguaje ensamblador. No piense que de esta forma le resultará fácil meterse dentro de los programas protegidos, ya que estos programas suelen ir también protegidos contra el desensamblaje. Podrá desensamblarlos, pero probablemente no le valdrá la pena el esfuerzo.

PUNTOS GENERALES SOBRE EL DEVPAC

Solamente he encontrado un par de problemas con el DEVPAC. Se los voy a contar por si se le presentan a usted, que no piense que está haciendo algo mal. He experimentado uno de ellos en dos máquinas diferentes, por lo que pienso que no es problema del 'hardware'. Si durante la impresión de un ensamblaje teclea CTRL/C un par de veces, el fichero de texto (mostrado con el comando "L") aparece corrompido.

El ensamblador DEVPAC parece que asume que siempre tiene papel en la impresora, y es un problema si no lo tiene. Otra cosa acerca de la impresora es que algunas convierten el signo # en el signo £. Esto puede causar cierta confusión si no se tiene en cuenta.

CONCLUSIONES

La programación en lenguaje máquina es muy divertida, pero también un poco irritante. Puede perder fácilmente el programa en el que está trabajando, si el mencionado programa hace fallar al ordenador, y tiene que apagar y encender de nuevo para poder seguir adelante. Por esta razón es bueno salvar el programa antes de intentar ejecutarlo, aunque haya hecho mínimas correcciones en él.

Una vez que esté usándolo, el DEVPAC es muy potente como herramienta de desarrollo de programas, su módulo de depuración le puede ser de gran ayuda en desenmascarar esos pequeños problemas que, aún el desarrollo más cuidadoso, no puede prever.

Cuando haya adquirido experiencia con el BASIC, intente hacer algo en ensamblador, estará más cerca de la máquina, y si puede usar cualquiera de los dos métodos de programación encontrará que muchos programas se pueden escribir mejor en una combinación de ambos lenguajes que solamente en BASIC. Además, si tiene cierta experiencia en programación en código máquina, estará en mejor posición para sus futuras actividades con los ordenadores

CAPITULO SEIS
Un Proyecto Avanzado

En el capítulo tres hemos visto algunos proyectos que podíamos construir por nosotros mismos, para incrementar la utilidad del ordenador CPC. En este capítulo veremos alguna aplicación más compleja para uno de ellos.

PROYECTO DE RED DE DATOS DE ÁREA LOCAL

El término "red de datos" se ha explicado de forma libre en los últimos años para definir las conexiones intercontinentales de ordenadores. En este capítulo describimos el CPCNET, que es una red de datos de bajo costo operada manualmente.

Este capítulo describe la forma de usar la 'hardware' del canal de trasferencia paralela, descrito en el capítulo tres (proyecto seis), para implementar una interconexión entre procesadores y permitirle conectar dos ordenadores CPC e intercambiar datos o programas entre ellos, a alta velocidad.

El CPCNET se ha desarrollado para su uso en lugares donde se usan varios CPC, y se necesita intercambiar entre ellos grandes cantidades de datos. En este esquema usted conecta simplemente un cable apropiado (detallado más adelante) entre ellos, y usa los comandos del CPCNET para comenzar la trasferencia. Esto será valioso cuando haya máquinas con cassette y con disco. Sin un medio de almacenamiento de datos común, la trasferencia de ficheros solo proporciona dolores de cabeza. Otro uso es la trasferencia de programas largos o grupos de datos a otra máquina, cuando se conjugan la alta velocidad con la trasferencia paralela, reducimos en gran medida el tiempo requerido cuando se compara con los métodos de trasmisión serie.

El CPCNET tiene los siguientes objetivos:

1) Costo lo más bajo posible.
2) Hacer posible la trasferencia de texto, versiones ASCII de programas o datos binarios entre dos máquinas.

El 'software' del CPC estará contenido en una EPROM. Necesitará la tarjeta de expansión de ROM conectada a su máquina (ver proyecto diez en el capítulo tres), pero no es necesario que tenga el programador de EPROM, ya que esperamos tener disponibles EPROMs grabadas - ver al principio de este libro.

LIMITACIONES Y FACILIDADES DEL CPCNET - EN DETALLE

Los nuevos comandos disponibles después de conectar la RON del CPCNET son:

|NET.OPEN Modifica los bloques de salto del CPC, de forma que al lanzar un comando LIST #8 desde el BASIC, mandará los caracteres ASCII

que forman el programa a través de la red de datos en lugar de a la impresora, mandando primero una cabecera (ver más adelante).

|NET.CLOSE Este comando restaura los bloques de salto a sus valores normales, de forma que el comando LIST #8 lista el fichero de nuevo a la impresora.

|NET.SEND,START,END Este comando manda el contenido de las direcciones de memoria en el rango especificado por START y END, a la máquina que recibe los datos.

|NET.REC Esta rutina espera a que comience la recepción de los datos. Después examina la cabecera (ver más adelante) y, si los datos que recibe son de otro tipo que no sea programas BASIC en versión ASCII, los carga en memoria empezando en la dirección que indica la cabecera. Tenga en cuenta que esto ocurrirá sin tener en cuenta los límites que haya puesto el BASIC. Si los datos recibidos son un programa BASIC en versión ASCII, se coloca en memoria como BASIC. (Vea la descripción de cómo se realiza esto en las notas del proyecto del disco en ROM en el capítulo tres - proyecto nueve). Una vez completa la carga, se deshacen las modificaciones a los bloques de salto.

Veamos las limitaciones del CPCNET;

Limitaciones

En un sistema de red de datos de bajo precio, las limitaciones son inevitables. El sistema mínimo que presentamos aquí cuenta con que el usuario hará las conexiones manualmente, conectando los finales del cable entre el sistema que manda y el que recibe. Hablando estrictamente, esto no es una red de datos, sino una interconexión de procesadores. Sin embargo, como usted puede trasferir ficheros BASIC a través de la conexión entre dos máquinas CPC en una oficina o un centro educativo, conectando simplemente el cable de conexión entre dos máquinas, Pienso que se la puede calificar de una red de datos operada manualmente. Si quiere desarrollar su propia caja negra de nodo de red de datos, puede usar este 'software' y 'hardware' como base para una red de datos multinodo.

Cabeceras de trasmisión

El CPCNET trasmite los datos como un simple flujo de datos sobre la conexión paralela. Si lo pensamos un poco descubriremos que la máquina que recibe debe ser avisada de cuándo termina la trasmisión. No lo podemos hacer mirando un carácter de fin de fichero, ya que en los ficheros binarios el carácter EOF es un código de máquina real, y puede aparecer en cualquier lugar. Esto crea la necesidad de mandar un pequeño número de octetos antes que los datos reales, que describen el tipo de los datos que se mandan y su tamaño. A esto se le llama cabecera y consta de 32 octetos. El formato de esta cabecera es el siguiente:

```
Octeto 1        :Tipo de datos. 0=binarios, 1=BASIC ASCII
Octeto 2        :Dirección de comienzo de datos (octeto inferior)
Octeto 3        :Dirección de comienzo de datos (octeto superior)
Octeto 4        :Dirección de fin de datos (octeto inferior)
Octeto 5        :Dirección de fin de datos (octeto superior)
Octetos 6-25    :Nombre del fichero (ver más adelante)
Octetos 26-32   :Reservados para uso futuro.
```

Los octetos de cabecera se mantienen en una memoria intermedia llamada - razonablemente - la memoria intermedia de cabecera.

DESCRIPCIÓN DETALLADA DEL 'SOFTWARE' DE LA ROM PARALELA DEL CPCNET

Estas notas asumen que usted tiene un conocimiento suficientemente bueno del contenido del manual de la ROM, SOFT 158. Si no lo tiene, debe saltar a la siguiente sección. En esta sección veremos del 'software' para el CPCNET, que se debe grabar en una EPROM que se montará en la tarjeta de expansión de ROM.

En varios puntos, el 'software del CPC funciona de forma similar al disco en ROM que se detalló en el capítulo tres. Ambos modifican las entradas del bloque de saltos para acceder al BASIC, pero el CPCNET va más allá en este punto y modifica los bloques de salto para permitir que el canal ocho del BASIC se convierta en canal de salida al 'hardware' del CPCNET. Ambos, el 'software del CPCNET y el del disco en ROM, están localizados en ROM montadas sobre la tarjeta de expansión de ROM. Y, finalmente, ambas usan una librería similar de subrutinas.

Hay cuatro módulos principales en el 'software del CPCNET. Estos se describen y listan, en el orden en que deben ser ensamblados. Después de que se hayan descrito todos los módulos y conexiones 'hardware', le daremos el listado del ensamblaje de la ROM terminada, junto con un vuelco hexadecimal de la misma.

1) **NETINIT**: Este es el nombre de las rutinas de inicialización del CPCNET. El listado de esta sección está en la figura 6.1: Las primeras líneas realizan la asignación de nombres a las direcciones de los registros del canal de trasferencia paralela (ver más adelante "Hardware del CPCNET"). Después viene la tabla de cabecera de la ROM paralela, tal como se especifica en el libro SOFT 158. Y por último la rutina NETGO - que es llamada automáticamente cuando la ROM del CPC detecta que está presente la ROM del CPCNET, durante la puesta en marcha de la máquina.

   NETGO reserva 256 octetos de memoria para uso exclusivo del CPCNET. La ROM del CPC no usará esta memoria. Limpia los indicadores de actividad. Todos los módulos principales del CPCNET tienen asignado un bit indicador en uno de los octetos de indicadores. Cuando entra la rutina pone el bit indicador, y lo limpia cuando termina. Esto sirve como ayuda cuando se intentan detectar los errores, ya que, si una rutina ha terminado en error, es posible determinar qué rutina estaba ejecutándose cuando ocurrió el final prematuro. Los octetos de indicadores le serán de gran utilidad si quiere modificar el 'software'. El acto final de NETGO es lanzar un mensaje a la pantalla para decirle que se ha inicializado el CPCNET. Este mensaje debe aparecer en cada encendido del CPC, siempre que tenga conectada la tarjeta de expansión de la ROM. Una vez inicializado NETGO, retorna a la ROM con el bit de acarreo a uno para indicar que todo ha ido bien.

```
 30 ; **         CPCNET     listado de la ROM                **
 40 ;
 50 ;
 60 ; Red de datos de comunicación entre ordenadores operada manualmente, para
 70 ; los ordenadores Amstrad CPC. Kernow computers (c) 1985
 80 ;
 90 ;
100 PTCSND:   EQU  #F8F0  ; Dirección registros PTC
110 PTCREC:   EQU  #F8F1
120 PTCCNT:   EQU  #F8F2  ; Señales control canales
130 PPICTL:   EQU  #F8F3  ; Registro control PTC chip PPI
140 ROMNUM:   EQU  5           ; Número zócalo ROM
150 ;
160 ;
170 ;
180           ORG  #C000
190 ;
200 ;
210           DEFB #1          ; ROM PARALELA
220           DEFB #1,#1,#1    ; Marca 1 Vers 1 rev 1
230           DEFW NAMTAB ; Apuntador a tabla de nombres
240           JP   NETGO  ; Rutina iniciación en encendido
250           JP   NETOPN
260           JP   NETCLOS
270           JP   NETSND
280           JP   NETREC
290 NAMTAB:   DEFM "NET INI"   ; Entrada iniciación
300           DEFB "T"+#80
310           DEFM "NET.OPE"
320           DEFB "N"+#80
330           DEFM "NET.CLOS"
340           DEFB "E"+#80
350           DEFM "NET.SEN"
360           DEFB "D"+#80
370           DEFM "NET.RE"
380           DEFB "C"+#80
390           DEFB 0           ; Fin tabla nombres
400 ;
410 ;
420 ; ** NETGO: Esta rutina inicializa la red de datos a tiempo de
430 ; ** encendido. Está limitada a preparar el PPI y crear
440 ; ** un área de trabajo en RAM que dejará libre
450 ; ** el CPC
460 NETGO:    PUSH AF
470           PUSH BC
480           DEC  H           ; HL contiene el límite superiordel área
490;                        de trabajo de 256 octetos
500           LD   BC,PPICTL   ; BC apunta al control del PPI
510           LD   A,#AF  ; Octeto de modo es AF
520           OUT  (C),A  ; escribirlo
530           LD   A,0
540           LD   (IY+#4E),A
550           LD   (IY+#4F),A  ; limpia octetos indicadores
560           PUSH IX
570           LD   IX,ININES   ; Decirle al usuario que está
580           CALL OUTPUT ; hecho con un mensaje
590           POP  IX
600           POP  BC
```

```
610           POP  AF
620           SCF
630           RET              ; Restaurar registros, poner acarreo
640 INIMES:   DEFB #0A,#0D
650           DEFM "** CPCNET  V1.1 iniciada **   "
660           DEFB #0A,#0D,#FF
670 ;
680 ;
690           JR   SNDEND
700 SUCESS:   DEFB #0A,#0D
710           DEFM "** Transfer. completa **"
720           DEFB #0A,#0D,#FF
730 SNDEND:   POP  BC
740           POP  IX
750           POP  HL
760           POP  DE
770           POP  AF
780           RET
790 ;
```

Fig 6.1

2) El siguiente módulo del 'software' del CPCNET se llama **NETOPEN**. Esta rutina abre el canal de trasferencia paralela como canal 8 del BASIC. El listado de NETOPEN se muestra en la figura 6.2. Después de lanzar un comando NET.OPEN, puede mandar programas BASIC o variables hacia el PTC (Canal de Trasferencia Paralela) usando sentencias como "LIST #8" o "PRINT #8,D$". Esto se realiza modificando las entradas del bloque de saltos que pertenecen a la impresora. Las entradas originales del bloque de saltos se salvan en el área de trabajo reservada por NETGO y se reemplazan por entradas que apuntan a las rutinas de mandar caracteres por el canal PTC.

```
 10 ; ** NETOPN:  Esta rutina del CPCNET abre el PTC como canal 8 **
 20 ; ** del BASIC, permitiendo la trasferencia sobre el CPCNET   **
 30 ; ** de versiones de programas BASIC en formato texto, usando **
 40 ; ** sentencias de tipo LIST #8 o PRINT #8                    **
 50 ;
 60 ;
 70 ;
 80 NETOPN:   PUSH AF
 90           PUSH HL
100           BIT  0,(IY+#4E)  ; Ver si bloques salto modificados
110           JR   NZ,MAKHDR   ; saltar esa sección
120           SET  0,(IY+#4E)  ; Si hay que modificarlos, ind. puesto
130           SET  0,(IY+#4F)  ; poner bit actividad
140           LD   HL,(#BD2B)  ; tomar dos octetos
150           LD   (IY),L ; salvar el primero
160           LD   (IY+#1),H   ; salvar el segundo
170           LD   HL,(#BD2D)  ; tomar dos último octetos
180           LD   (IY+#2),L   ; salvar el tercero
190           LD   (IY+#3),H   ; y el último
200           LD   A,#DF  ; cargar instrucción rearranque
210           LD   (#BD2B),A   ; en lugar de lo que contenía
220           LD   HL,SEND     ; HL apunta a rutina SEND
```

```
230             LD   (IY+#10),L  ; Poner primer octeto
240             LD   (IY+#11),H  ; Y el segundo
250             LD   H,ROMNUM    ; Ahora número ROM
260             LD   (IY+#12),H  ; Número ROM en dirección alta
270             PUSH IY          ; Copia IY en HL
280             POP  HL
290             PUSH DE          ; salvar DE
300             LD   DE,#10
310             ADD  HL,DE  ; calcula direc. bloque salto
320             POP  DE          ; restaurar DE
330             LD   (#BD2C),HL  ; colocar direc. bloque salto
340             LD   A,#C9  ; cargar inst. RET en la entrada
350             LD   (#BD2E),A   ; después de PRINT CHAR
360 MAKHDR:     LD   A,#1   ; construir mem. int. cabecera
370             LD   (IY+#50),A  ; poner tipo datos (BASIC=01)
380             LD   A,0         ; cero a direcciones mem. int.
390             LD   (IY+#51,A)
400             LD   (IY+#52,A)
410             LD   (IY+#53,A)
420             LD   (IY+#54,A)  ; cero a campos datos
430             PUSH IY          ; Copiar IY a HL
440             POP  HL
450             LD   BC,#55 ; HL apunta a campo nombre
460             ADD  HL,BC
470             LD   B,16   ; 16 caracteres a cero
480 OPN009:     LD   (HL),A ; cero un octeto
490             INC  HL          ; incrementa apuntador
500             DJNZ OPN009 ; repite
510             CALL SNDHDR ; Mandar cabecera
520             LD   IX,OPEND    ; Decirselo al usuario
530             CALL OUTPUT
540             JP   OPNEND ; y terminar
550 OPEND:      DEFB #0A,#0D
560             DEFM "** CPCNET esta abierto para mandar programas  **"
570             DEFB #0A,#0D
580             DEFM "** o variables BASIC por el canal 8. Puede    **"
590             DEFB #0A,#0D
600             DEFM "** usar sentencias LIST #8 o PRINT #8 para hacerlo.    **"
610             DEFB #0A,#0D
620             DEFM "** cuando haya terminado, ejecute un comando           **"
630             DEFB #0A,#0D
640             DEFM "** |NET.CLOSE para terminar la trasmision.             **"
650             DEFB #0A,#0D,#FF
660 OPNEND:     POP  HL
670             POP  AF          ; restaurar
680             RES  0,(IY+#4F)  ; quitar indic. actividad
690             RET              ; y volver
```

Fig 6.2

Debido a que usamos direcciones remotas, es esencial que sea correcta la asignación de ROMNUM del zócalo de ROM donde ha montado su ROM del CPCNET. Por ejemplo, si está montada en el zócalo 4, debe alterar el valor de ROMNUM que aparece en la figura 6.1, y alterar la versión grabada dentro de la EPROM. El valor por defecto de ROMNUM debe ser válido para la mayoria de los sistemas.

Una vez que se han modificado los bloques de salto, la siguiente acción de NETOPEN es preparar una cabecera válida, en la memoria intermedia que está localizada en el área de trabajo de CPCNET. Nota: A la entrada de las rutinas CPCNET, la ROM del CPC ha puesto el registro IY previamente como dirección base del área de trabajo del CPCNET. Como en el disco en ROM, los octetos del área de trabajo son referidos como (IY+#n), donde n es el número que debe usar como como desplazamiento desde el valor de IY, para forma la dirección deseada.

Cuando casi se ha completado, NETOPEN imprime un mensaje para decirle al usuario como debe usar el CPCNET para mandar datos por el canal #8. Al igual que las otras rutinas principales, NETOPEN pone el indicador de actividad mientras se está ejecutando, pero también pone el bit (IY+#4E) a cero para indicar que ha modificado las entradas del bloque de saltos de la impresora. La rutina NETCLOSE usa posteriormente este indicador.

3) A continuación viene la rutina **NETCLOSE**. Esta rutina realiza la tarea opuesta a NETOPEN. Examina los indicadores que pusieron NETOPEN y NETREC (ver más adelante) cuando modificaron los bloques de salto. Si está puesto alguno de los indicadores, son restauradas desde el área de trabajo del CPCNET, las entradas correspondientes del bloque de saltos, con sus valores originales, devolviendo al CPC a su estado original. En cualquier caso, NETCLOSE manda siempre un octeto Hex FF al canal del PTC para indicar al BASIC que ha finalizado la trasmisión. La figura 6.3 es un listado de NETCLOSE.

```
10  ;
20  ;
30  ;
40  ; ** NETCLO: Esta rutina del CPCNET cierra el canal PTC al BASIC
50  ; ** para entrada y salida. Restaura a sus valores originales las
60  ; ** entradas de los bloques de salto que había modificado
70  ; ** previamente. Si no lo hay ninguno modificado, no toma ninguna
80  ; ** acción.
90  ;
100 NETCLO:   PUSH HL
110           PUSH AF
120           SET  1,(IY+#4E)  ; Poner bit actividad octeto 2
130           BIT  0,(IY+#4E)  ; ver si están cambiadas entradas impres.
140           JR   Z,CLS001    ; no, saltar
150           RES  0,(IY+#4E)  ; limpiar indicador
160           LD   L,(IY) ; tomar primer octeto
170           LD   H,(IY+#1)   ; tomar segundo
180           LD   (#BD2B),HL  ; restaurarlos
190           LD   L,(IY+#2)   ; tomar tercer octeto
200           LD   H,(IY+#3)   ; tomar último octeto
210           LD   (#BD2D),HL  ; restaurarlos
220 CLS001:   BIT  7,(IY+#4F)  ; Ver si se han modificado las
230           JR   Z,CLSEND    ; entradas CAS xx, no, saltar
240           RES  7,(IY+#4F)  ; limpiar indicador. Vamos a
250           LD   L,(IY+#4)   ; restaurarlas. Primer octeto
260           LD   H,(IY+#5)   ; segundo octeto
270           LD   (#BC77),HL  ; restaurarlos
280           LD   L,(IY+#6)   ; tercer octeto
290           LD   H,(IY+#7)   ; cuarto octeto
300           LD   (#BC79),HL  ; restaurarlos
```

```
310           LD   L,(IY+#8)   ; ahora la entradas de CAS_IN_CHAR
320           LD   H,(IY+#9)   ; restaurarlas
330           LD   (#BC80),HL
340           LD   L,(IY+#A)
350           LD   H,(IY+#B)
360           LD   (#BC82),HL  ; Hecho
370 CLSEND:   POP  AF
380           POP  HL
390           RES  1,(IY+#4E)  ; restaurar bit actividad
400           RET              ; y retornar
```

Fig 6.3

4) A continuación viene **NETSEND**. Esta rutina realiza la trasmisión de los datos entre dos direcciones de memoria especificadas. La primera es la dirección de comienzo y la segunda es la de terminación. NETSEND manda sobre el PTC una copia del contenido de esas posiciones de memoria. En la figura 6.4 está listada la rutina NETSEND. Usted debe especificar las direcciones de comienzo y fin, si no lo hace, o si la dirección de fin es menor que la de principio, le lanza un mensaje. Los valores de estas direcciones se mandan en la cabecera de forma que la máquina que lo recibe sepa en qué parte de la memoria debe poner los datos. Cuando se han mandado todos los octetos dentro del rango de direcciones, se imprime en la pantalla del CPC el mensaje "** Transfer. completa **". Si por alguna razón se corta la trasmisión, puede abortarla tecleando CTRL/A en el teclado de su CPC.

```
 10 ;
 20 ; ** NET SEND: Se invoca con un comando como
 30 ; ** |NET.SEND,DIR_PRINC,DIR_FIN
 40 ; ** La trasmisión se realiza con los octetos comprendidos
 50 ; ** entre las direcciones de principio y de fin
 60 ;
 70 ;
 80 NETSND:   PUSH AF
 90           PUSH DE
100           PUSH HL
110           PUSH IX
120           PUSH BC
130           CP   #2          ; Ver si ha pasado dos parametros
140           JR   Z,SND001    ; Si, saltar
150 SND004:   LD   IX,BUMPAR   ; No, msg error
160           CALL OUTPUT ; lanzar msg
170           JP   SNDEND ; y adios
180 BUMPAR:   DEFM "** Debe especificar direcciones   **"
190           DEFB #0D,#0A
200           DEFM "** de principio y final.  Eg-|NET.SEND,1000,2000 **"
210           DEFB #0A,#0D,#FF
220 SND001:   PUSH IY          ; Copia IY a HL
230           POP  HL
240           LD   BC,#50
250           ADD  HL,BC  ; HL apunta mem. int. cabecera
260           LD   (HL),#00    ; Tipo datos es binario
```

```
270             INC  HL          ; HL=siguiente octeto
280             LD   A,(IX+2)    ; Comienzo datos (L)
290             LD   (HL),A ; ponerlo en mem. int.
300             LD   A,(IX+3)    ; Comienzo datos (H)
310             INC  HL          ; Incrementa apuntador
320             LD   (HL),A ; ponerlo en mem. int.
330             LD   A,(IX) ; Fin datos (L)
340             INC  HL
350             LD   (HL),A ; ponerlo en mem. int.
360             LD   A,(IX+1)    ; fin datos (H)
370             INC  HL          ; incrementar apuntador
380             LD   (HL),A ; ponerlo en mem. int.
390             INC  HL          ; incrementar apuntador
400             LD   B,16   ; 16 espacios en cabecera
410             LD   A,#20
420 SND002:     LD   (HL),A
430             INC  HL
440             DJNZ SND002 ; repetir 16 veces
450             CALL SNDHDR ; mandar cabecera
460             LD   L,(IY+#51)
470             LD   H,(IY+#52)  ; HL apunta princ. datos
480             LD   C,(IY+#53)  ; y BC a fin datos
490             LD   B,(IY+#54)
500             SBC  HL,BC  ; ver si fin > principio
510             JP   NC,SND004   ; si, saltar
520             LD   L,(IY+#51)  ; reacrgar dir. pricipio
530             LD   H,(IY+#52)
540 SND003:     LD   A,(HL) ; Tomar un octeto de memoria
550             CALL SEND   ; mandarlo
560             JR   C,SNDEND    ; Terminar si se pulsó CTRL/A
570             INC  HL          ; incrementar apuntador
580             LD   A,L         ; copia L en A
590             CP   (IY+#53)    ; Fin del área (octeto inferior)
600             JR   NZ,SND003   ; no, saltar
610             LD   A,H         ; tomar octeto superior
620             CP   (IY+#54)    ; ver si coincide
630             JR   NZ,SND003   ; no, saltar
640 ; ** Cuando llegamos aquí ya se ha terminado la trasferencia
650 ; ** así que manda los mensajes correspondientes
660 ;
670             LD   IX,SUCESS
680             CALL OUTPUT
```

Fig 6.4

5) La última de las rutinas principales es **NETREC**. Este es el módulo principal de recepción. Cuando usted lanza el comando NET.REC, la rutina comienza esperando la recepción de un octeto marcador de cabecera en el canal de recepción del PTC, usando la rutina de NETLIB, RECHDR. Si por alguna razón no se recibe la cabecera, puede salir de RECHDR con CTRL/A; cuando RECHDR recibe la cabecera, retorna a NETREC. Como se puede ver en el listado de la rutina NETREC en la figura 6.5, lo siguiente que hace es poner el bit de actividad en el octeto de función situado en el área de trabajo del CPCNET. Después examina el octeto de tipo de datos de la cabecera recibida.

```
10  ;
20  ; ** NETREC: Esta rutina de CPCNET es el receptor principal  **
30  ; ** de datos desde el canal receptor del PTC. Recibe la     **
40  ; ** cabecera empaquetada y mira si lo que está recibiendo   **
50  ; ** es un programa BASIC. Si lo es modifica los bloques de  **
60  ; ** salto correspondientes y lanza un mensaje para que      **
70  ; ** introduzca un comando LOAD. Si el tipo de datos         **
80  ; ** recibido es binario, NETREC realiza por si mismo el     **
90  ; ** trabajo de almacenamiento de los datos recibidos.       **
100 ; ** NOTA: cuando recibe datos binarios no comprueba si el   **
110 ; ** área usada para almacenarlos está libre o no.           **
120 ;
130 ;
140 NETREC:   PUSH AF
150           PUSH HL
160           PUSH BC
170           PUSH DE          ; Usamos todos los registros
180           CALL RECHDR ; Recibir cabecera
190           JP   C,RECEND    ; Abortar si CTRL/A
200           SET  2,(IY+#4F)  ; poner bit de actividad
210           LD   A,(IY+#50)  ; obtener octeto de tipo
220           CP   #01         ; ver si es BASIC
230           JR   Z,BASREC    ; Si, saltar a receptor BASIC
240 BINREC:   LD   L,(IY+#51)  ; HL=dirección principio
250           LD   H,(IY+#52)
260           LD   E,(IY+#53)  ; DE=dirección fin
270           LD   D,(IY+#54)
280 REC001:   CALL RECEEV ; Tomar un carácter
290           JP   C,RECEND    ; abortar si CTRL/A
300           LD   A,(HL) ; almacenar el octeto
310           LD   A,E         ; ver si el octeto inf. del apunt.
320           CP   L           ; coincide con el fin
330           JR   NZ,REC002   ; no, saltar
340           LD   A,D         ; comparar octetos altos
350           CP   H           ; ver si coinciden
360           JP   Z,RECEND    ; si, saltar
370 REC007:   INC  HL          ; incrementar apuntador
380           JR   REC001 ; y repetir
390 BASREC:   BIT  0,(IY+#4E)  ; ver si está modificado
400           JR   NZ,REC003   ; si, saltar esta sección
410           LD   A,(#BC77)   ; cargador BASIC
420           LD   (IY+#4),A   ; salvar primer octeto CAS_IN_OPEN
430           LD   HL,(#BC78)  ; tomar los otros dos
440           LD   (IY+#5),L   ; salvarlos en área trabajo
450           LD   (IY+#6),H
460           LD   A,ROMNUM    ; A=número ROM
470           LD   (IY+#15),A  ; es tercer octeto dir. remota
480           PUSH IY
490           POP  HL          ; copiar IY en HL
500           LD   DE,#13 ; HL=dirección remota
510           ADD  HL,DE
520           LD   (#BC78),HL  ; poner bloque de salto
530           LD   A,#DF  ; y el RST 18
540           LD   (#BC77),A   ; también
550           LD   A,(#BC7A)   ; Salvar el siguiente octeto
560           LD   (IY+#7),A   ; Poner un RET
570           LD   A,#C9
580           LD   (#BC7A),A
```

```
590           LD   DE,BAS001   ; DE apunta a BAS001
600           LD   (HL),E ; colocarla en dirección remota
610           INC  HL
620           LD   (HL),D
630           LD   A,(#BC80)   ; Modificar CAS_IN_CHAR
640           LD   (IY+#8),A   ; salvar octetos originales
650           LD   HL,(#BC81)
660           LD   (IY+#9),L
670           LD   (IY+#A),H
680           LD   A,(#BC83)   ; También el tercer octeto
690           LD   (IY+#B),A
700           LD   A,#C9  ; poner aquí un RET
710           LD   (#BC83),A
720           LD   HL,RECBAS
730           LD   (IY+#16),H  ; Reemplazar la dirección
740           LD   (IY+#17),L  ; remota de la CAS_IN_CHAR
750           LD   A,ROMNUN    ; y ROMNUM completa dirección remota
760           LD   (IY+#18),A
770           PUSH IY
780           POP  HL          ; copiar IY en HL
790           LD   BC,#16 ; BC=despl. a dir. remota
800           ADD  HL,BC  ; HL=dirección remota
810           LD   A,#DF  ; RST 18 en bloque salto
820           LD   (#BC80),A   ; entrada para CAS_IN_CHAR
830           LD   (#BC81),HL  ; y después dir. remota
840           SET  7,(IY+#4F)  ; poner bit especial en octeto
850  ; función, indica cambio entradas bloque salto E/S de cassette
860  RECO03:  LD   IX,RECMES   ; Decimos como recibir
870           CALL OUTPUT
880           JP   RECEND
890  RECMES:  DEFB #0A,#0D
900           DEFM "** Los datos que llegan se van a meter en el BASIC.  **"
910           DEFB #0A,#0D
920           DEFM "** Teclee los comandos LOAD o MERGE para empezar     **"
930           DEFB #0A,#0D
940           DEFM "** la recepción.              CPCNET                 **"
950           DEFB #0A,#0D,#FF
960  RECEND:  POP  DE          ;
970           POP  BC
980           POP  HL
990           POP  AF
1000          RES  2,(IY+#4F)  ; Quitar bit de actividad
1010          RET              ; y retornar
1020 ;
```

Fig 6.5

El tipo de datos encontrado en la cabecera, determina cual de los dos caminos de la rutina se debe tomar. Si indica que los datos que llegan son BASIC, se comienza una secuencia más compleja, esta se describirá separadamente dentro de un momento. Si el tipo de datos es binario, empieza una pequeña subrutina que pone la dirección de principio de la cabecera como su dirección de principio en memoria, y la dirección de fin de la cabecera como su dirección más alta de almacenamiento. Los octetos se cargan según se reciben por el canal de recepción del PTC, incrementando el apuntador cada vez que se almacena un octeto. La recepción termina cuando el apuntador llega a tener el valor de la dirección de fin.

Hay una implicación importante en la recepción de datos binarios de este modo. Si las direcciones de principio y fin son erróneas, por ejemplo 0000 a FFFF, el CPC puede fallar fácilmente porque se machacan las variables del sistema en RAM. No es posible una protección contra esto sin limitar su utilidad.

Si el octeto de tipo de datos indica que son BASIC, el usuario requiere la cooperación del sistema operativo. Lo que sucede realmente es que NETREC salva el contenido de las entradas de los bloques de salto de las rutinas CAS_IN_OPEN y CAS_IN_CHAR y las reemplaza con entradas que apuntan a las rutinas de la librería NETLIB de subrutinas. Estas son BAS001 y RECBAS. Estas dos rutinas suministran al BASIC las mismas entradas de las subrutinas a las que han reemplazado. BAS001 duplica la función de CAS_IN_OPEN haciendo creer al BASIC que está abriendo ficheros en cassette o disco. Como dijimos en el capítulo tres, el único parámetro pasado al BASIC que se comprueba es el tipo de fichero. BAS001 pasa la BASIC un tipo de fichero ASCII. RECBAS reemplaza CAS_IN_CHAR, que pasa al BASIC, carácter a carácter, el contenido del fichero abierto, pasando condiciones de indicador específicas y un EOF (Hex 1B) cuando se han suministrado todos los caracteres del fichero.

Una vez que se han modificado todas las entradas del bloque de saltos, se manda un mensaje auto explicativo para que el usuario teclee el comando LOAD. El BASIC comenzará a cargar los caracteres que vienen por el canal de recepción del PTC por medio de las rutinas BAS001 y RECBAS. Para terminar la transmisión, la máquina debe mandar un octeto con el séptimo bit puesto a uno. Cuando sucede esto, la recepción termina y el programa BASIC está cargado. En cualquier momento durante la carga, usted puede teclear CTRL/A para abortarla. El BASIC recibirá una condición de BREAK, como si usted hubiera pulsado la tecla ESC.

6) El listado de **NETLIB** aparece en la figura 6.6. Esta figura muestra las distintas subrutinas usadas por las rutinas principales de CPCNET. Algunas de estas son familiares del disco en ROM, pero todas tienen notas explicativas de lo que hacen. La rutina SEND puede necesitar alguna explicación posterior, y cómo trata directamente el PTC, debe estudiar también las notas de 'hardware' que siguen. SEND es usado por cualquier rutina que desee mandar un octeto por el CPCNET. SEND espera a tener libre la trasmisión para aceptar el octeto. Si mientras SEND está esperando, detecta que usted ha tecleado algo en el teclado del CPC, la operación es interrumpida y retorna a la rutina que la ha llamado, con el bit de acarreo puesto, para indicar que ha terminado anormalmente. Como ya dijimos, los comentarios para las otras subrutinas de NETLIB explican suficientemente su propósito.

```
 90 ; ------------------------------------------------------------------------
 91 ;              NETLIB CPCNET Librería de subrutinas
 92 ; ------------------------------------------------------------------------
100 ;
110 ;
120 ; ** SNDHDR: Esta subrutina manda el contenido    **
130 ; ** a la memoria intermedia de cabecera al PTC.  **
```

```
140 ; ** IY apunta a la base del área de trabajo      **
150 ;
160 SNDHDR:   PUSH AF
170           PUSH HL
180           PUSH BC
190           SET  4,(IY+#4F)  ; Poner bit actividad
200           LD   A,#AA
210           CALL SEND   ; mandar octeto marcador
220           JR   C,HDR001    ; saltar si tecla pulsada
230           PUSH IY
240           POP  HL          ; Copiar IY en Hl
250           LD   BC,#50
260           ADD  HL,BC  ; HL apunta a mem. int. cabecera
270           LD   B,#30  ; B=contador bucle
280 SNDLOP:   LD   A,(HL) ; tomar un octeto
290           CALL SEND   ; mandarlo
300           JR   C,HDR001    ; salir si tecla pulsada
310           INC  HL          ; incrementar apuntador
320           DJNZ SNDLOP ; repetir bucle
330           LD   A,0         ; limpiar acarreo al final
340           ADC  0
350 HDR001:   POP  BC
360           POP  HL
370           POP  AF
380           RES  4,(IY+#4F)  ; Limpiar bit actividad
390           RET              ; retornar
400 ;
410 ; ** La rutina de salida manda mensajes a la pantalla.   **
420 ; ** IX apunta al principio del mensaje, que termina con **
430 ; ** un octeo de valor #FF. AF, IX y HL corrompidos      **
440 OUTPUT:   LD   A,(IX) ; Toma un octeto
450           BIT  7,A         ; ver si es EOB
460           RET  NZ          ; si, retornar
470           CALL #BB5A  ; mandar el carácter
480           INC  IX          ; incrementar apuntador
490           JR   OUTPUT ; y repetir
500 HEXAS:    LD   H,A         ; salvar valor original
510           AND  #0F         ; dejar mitad inferior
520           OR   #30         ; convertir en ASCII
530           CP   #3A         ; ver si es alfabético
540           JP   M,HIGH ; no, saltar
550           ADD  A,#7   ; convertir en hex
560 HIGH:     LD   L,A         ; Poner dígito inferior en L
570           LD   A,H         ; restaurar original
580           SRL  A           ; poner mitad superior en inferior
590           SRL  A
600           SRL  A
610           SRL  A
620           OR   #30
630           CP   #3A         ; ver si es alfabético
640           JP   M,DONE ; no, terminado
650           ADD  A,#7   ; ajustar como antes
660 DONE:     LD   H,A         ; salvar en H
670           RET              ; y terminar
680 ;
690 ;
700 ; ** BAS001: Esta rutina reemplaza a la rutina CAS_IN_OPEN de la
710 ; ** ROM durante la recepción de programas BASIC desde el CPCNET
720 ; ** Lo carga dentro del BASIC como si hubiera abierto un fichero
```

```
730 ; ** con un programa BASIC salvado en ASCII
740 ;
750 ;
760 BAS001:   LD   A,#16  ; Carga código tipo ASCII
770           CP   #0           ; limpia indicador Z
780           SCF               ; pone indicador acarreo
790           RET               ; vuelve al BASIC
800 ;
810 ;
820 ;
830 ;
840 ; ** RECBAS: Esta rutina reemplaza a la rutina CAS_IN_CHAR de la
850 ; ** ROM, que introduce programas BASIC tomados del cassette o
860 ; ** disco, carácter a carácter. Esta rutina usa la subrutina
870 ; ** RECEEV e indica el fin de fichero al BASIC si el octeto
880 ; ** recibido tiene puesto el bit siete.
890 ;
900 ;
910 ;
920 RECBAS:   CALL RECEEV ; Toma un carácter
930           JR   NC,RECB01   ; Salta si no CTRL/A
940           LD   A,0         ; si lo ha tecleado
950           ADC  A,#0   ; pone los bits de cero y acarreo
960           RET               ; pasa BREAK al BASIC y retorna
970 RECB01:   BIT  7,A          ; ver si bit 7 puesto
980           JR   Z,RECB02     ; no, salta
990           LD   A,0
1000          ADC  A,#0
1010          LD   A,#1F  ; limpia bits cero y acarreo
1020          RET               ; A=EOF y retorna
1030 RECB02:  CP   #FF          ; limpiar indicador Z
1040          SCF               ; poner acarreo para decirlo
1050          RET               ; al BASIC que OK y reotrnar
1060 ;
1070 ;
1080 ; ** RECEEV: Esta rutina retorna a otras rutinas los caracteres
1090 ; ** que recibe por el canal de trasferencia paralela CPCNET.
1100 ; ** espera a que se ponga en estado lógico alto la señal IBF
1110 ; ** (mem. int. de entrada llena) de la puerta B, que indica
1120 ; ** que está disponible el octeto. Mientras espera, llama a
1130 ; ** la rutina de la ROM que comprueba si se ha tecleado CTRL/A
1140 ; ** Si se ha hecho, retorna con el indicador de acarreo puesto.
1150 ; ** Cuando lee un octeto, lo pasa dentro del registro A.
1160 ;
1170 ;
1180 ;
1190 RECEEV:   PUSH BC
1200           SET  5,(IY+#4F)  ; Comprobar bit actividad
1210 RECEE2:   LD   BC,PTCCNT   ; BC apunta al canal de control
1220           IN   A,(C)  ; Obtener estado control
1230           BIT  1,A         ; ver si hay IBF
1240           JR   NZ,RECEE1   ; si, saltar
1250           CALL #BB09  ; comprobar CTRL/A
1260           JR   NC,RECEE2   ; no, repetir bucle
1270           CP   #1          ; ver si es CTRL/A
1280           JR   NZ,RECEE2   ; no, repetir bucle
1290           SCF              ; poner acarreo
1300           JR   RECOUT ; e irse
1310 RECEE1:   LD   BC,PTCREC   ; BC apunta a canal recepción
```

```
1320            IN   A,(C)  ; tomas el carácter
1330            CP   A          ; limpiar acarreo
1340  RECOUT:   POP  BC         ; limpiar pila
1350            RES  5,(IY+#4F) ; quitar bit actividad
1360            RET             ; y volver
1370  ;
1380  ;
1390  ; ** SEND: Esta rutina de CPCNET manda al canal de trasmisión    **
1400  ; ** del PTC, el carácter que le llega en el registro A, pero    **
1410  ; ** primero espera a que el canal está disponibles - indicado   **
1420  ; ** por el bit 7 alto en la puerta C - PTCCNT.                  **
1430  ; ** Si el usuario pulsa una tecla mientras SEND está esperando, **
1440  ; ** pone el inidicador de acarreo y retorna                     **
1450  ;
1460  SEND:     PUSH BC
1470            PUSH HL
1480            SET  6,(IY+#4F) ; poner bit actividad
1490  SEND00:   LD   BC,PTCCNT  ; BC apunta al registro de estado del PTC
1500            IN   L,(C)  ; L=estado
1510            BIT  7,L        ; ver canal trasmisión listo
1520            JR   NZ,SEND01  ; si, saltar
1530            PUSH AF         ; salvar el octeto a mandar
1540            CALL #BB09  ; ver si se ha tecleado algo
1550            JR   NC,SEND02  ; no, saltar
1560            POP  AF         ; recuperar AF
1570            SCF             ; poner acarreo
1580            JR   SEND04 ; y terminar
1590  SEND02:   POP  AF         ; Si no se ha tecleado, recuperar AF
1600            JR   SEND00 ; y repetir bucle
1610  SEND01:   LD   BC,PTCSND  ; BC apunta a canal trasmisión
1620            OUT  (C),A  ; y manda un octeto
1630            AND  A          ; limpia acarreo
1640  SEND04:   POP  HL         ; restaura la pila
1650            POP  BC
1660            RES  6,(IY+#4F) ; limpia bit actividad
1670            RET             ; retorna
1680  ;
1690  ;
1700  ;
1710  ;
1720  ; ** RECHDR: Esta rutina recive una cabecera del CPCNET, desde
1730  ; ** el canal de recepción del PTC. La cabecera debe tener el
1740  ; ** formato correcto, o mandará un mensaje
1750  ;
1760  ;
1770  RECHDR:   PUSH BC
1780            PUSH HL
1790            PUSH DE
1800            SET  3,(IY+#4F) ; poner bit actividad
1810            PUSH IY         ; copia IY a HL
1820            POP  HL
1830            LD   BC,#50
1840            ADD  HL,BC  ; HL apunta a mem. int. cabecera
1850  RECH01:   CALL RECEEV ; toma un octeto
1860            JP   C,RECH04   ; salir si lo quiere el usuario
1870            CP   #AA        ; ver si es marcador principio
1880            JR   Z,RECH01   ; no, repetir bucle
1890            CALL RECEEV ; si, comienza recepción
1900            LD   E,A        ; validar tipo datos (solo 0 o 1)
```

```
1910            AND  #FE          ; enmascarar bit ilegales
1920            JR   Z,RECH02     ; si no hay bits puestos, saltar
1930            LD   IX,RECHER    ; los hay, lanzar mensaje
1940            CALL OUTPUT
1950            JP   RECH04 ; y abortar recepción
1960  RECHER:   DEFB #0A,#0D
1970            DEFM "** Atencion: RECHDR detecto el tipo de datos ilegal **"
1980            DEFB #0A,#0D
1990            DEFM "** recepcion cabecera abortada - avisar al emisor   **"
2000            DEFB #0A,#0D,#FF
2010  RECH02:   LD   (HL),E ; tipo datos en mem. int.
2020            INC  HL           ; incrementar apuntador
2030            LD   B,#1B  ; 20 octetos más para cabecera
2040  RECH03:   CALL RECEEV ; tomar un octeto
2050            JR   C,RECH04     ; abortar si lo quiere el usuario
2060            LD   (HL),A ; almacenar octeto
2070            INC  HL           ; actualizar apuntador
2080            DJNZ RECH03 ; repetir hasta terminar
2090  RECH04:   POP  DE
2100            POP  HL
2110            POP  BC           ; restaurar registros
2120            RES  3,(IY+#4F)  ; limpiar bit actividad
2130            RET               ; retornar
```

Fig 6.6

EL 'HARDWARE' DEL CPCNET

El 'hardware' del CPCNET se revisó ya en el capitulo tres como canal de trasferencia paralela (PTC). Ahora vamos a ver los detalles de uso en la aplicación CPCNET. La figura 6.7 muestra los detalles del cable de interconexión requerido para conectar dos máquinas CPC provistas con un PTC para poder usar la trasferencia de datos por medio del CPCNET.

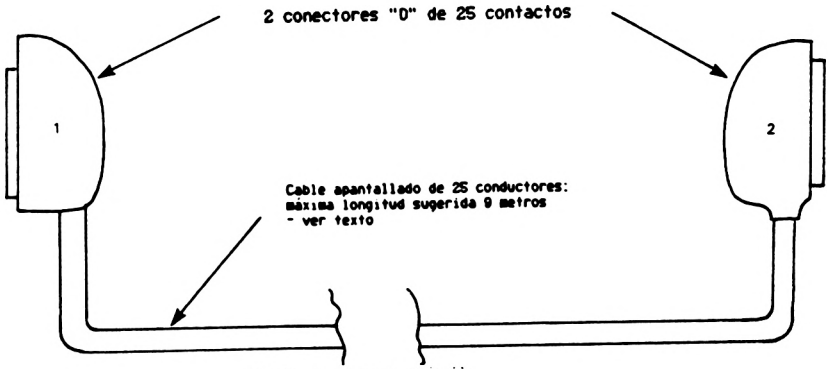
Los chips PPI usados en el diseño tienen la capacidad de realizar trasmisiones de octetos simples, independientemente del ordenador. Esta capacidad fue descrita el el capítulo tres. La extensión de la que se hace cargo el 'hardware' del PPI nos permite usar la puerta C como registro de estado para el PTC. Los bits de registro de estado se usan para las siguientes funciones:

Nombre	TCRDY	ACK	XXX	XXX	XXX	STB	RCRDY	XXX
bit	7	6	5	4	3	2	1	0

0/3/4/5 = Bits sin usar (designados como XXX)

7 TCRDY = Cuando lo lee como lógico uno, significa que el canal de trasmisión (PTCSND en el listado del ensamblaje) está preparado para recibir un octeto.

Fig 6.7 Cable de conexión del CPCNET



Cuando use el cable indicado debe realizar las siguientes conexiones para interconectar el CPCNET

CON #1	Color cable	CON #2
1	Rojo - blindado	1
2	Azul	10
3	Verde	11
4	Amarillo	12
5	Blanco	13
6	Negro	14
7	Marrón	15
8	Violeta	16
9	Naranja	17
10	Rosa	2
11	Turquesa	3
12	Gris	4
13	Rojo/Azul	5
14	Verde/Rojo	6
15	Amarillo/Rojo	7
16	Blanco/Rojo	8
17	Rojo/Negro	9
18	Rojo/Marrón	21
19	Amarillo/Azul	20
20	Blanco/Azul	19
21	Azul/Negro	18
25	Naranja/Verde	25

6 ACK = Este bit, con un impulso bajo en la máquina receptora (al otro lado de la línea), acepta el octeto que le acaban de mandar.

2 STB = Se pone a nivel bajo cuando recibe un sincronismo de la máquina remota, dentro del registro de recepción del PTC.

1 RCDY = Preparado el canal de recepción. Cuando está alto, esta señal indica que hay un octeto disponible en el registro de recepción del PTC.

El 'hardware' del PTC para conectarse con otros dispositivos que no sean el PTC, es muy simple. Por ejemplo, mi equipo de prueba para el PTC tiene una conexión con un UART. Los lectores no deben encontrar

dificultad para adaptar el 'hardware' para su conexión con otras máquinas.

APLICACIONES DEL CPCNET

El CPCNET se puede adaptar para transferir casi todos los tipos de datos desde una máquina a otra. Su principal ventaja es la velocidad a la que se puede realizar la trasmisión. Por ejemplo, para trasferir 16K octetos (como una imagen de pantalla) tarda poco menos de dos segundos. La velocidad a la que se efectúa la trasmisión es despreciable comparada con el tiempo necesario para poder comenzar la transmisiòn, pero no tiene importancia si mantenemos dos CPC provistos de PTC, conectados permanentemente.

LISTADO DE LA ROM DEL CPCNET

La figura 6.8 es el listado del ensamblaje y el vuelco hexadecimal del 'software' del CPCNET que se ha de grabar en la ROM. El programa está muy comentado, por lo que no debe tener problemas si intenta hacer alguna modificación para acoplarlo a sus propias necesidades. El área de trabajo del CPCNET (a la que apunta IY cuando se entra en cualquier rutina del CPCNET) se usa de la siguiente forma:

IY+0 a IY+3 = Usada para salvar la entrada del bloque de saltos de la rutina MC_PRINT_CHAR y el octeto que la sigue.

IY+4 a IY+7 = Usada para salvar el contenido original de la entrada del bloque de salto para CAS_IN_OPEN y el octeto que la sigue.

IY+#8 a IY+#B = Usada para salvar la entrada original del bloque de salto para CAS_IN_CHAR.

IY+#10 a IY+#12 = Contiene la dirección remota de la rutina SEND.

IY+#13 a IY+#15 = Contiene la dirección remota de la rutina BASOO1.

IY+#16 a IY+#19 = Contiene la dirección remota de la rutina RECBAS.

IY+#4E = Octeto 2 de indicadores. Si está puesto el bit cero, significa que las entradas del bloque de saltos de la impresora se han modificado. Si está puesto el bit uno, significa que NETCLOS está activa.

IY+#4F = Octeto 1 de indicadores.
BIT 0 puesto = NETOPEN activa
BIT 2 puesto = NETREC activa
BIT 3 puesto = RECHDR activa
BIT 4 puesto = SNDHDR activa
BIT 5 puesto = RECEEV activa
BIT 6 puesto = SEND activa

```
                                BIT 7 puesto = NETOPEN activa
                                BIT 7 puesto significa que las entradas
                                        del bloque de salto de las ru-
                                        tinas del cassette se han mo-
                                        dificado por el CPCNET.

IY+#50 a IY+#7F =               Memoria intermedia de cabecera - especí-
                                ficamente:

IY+#50 =                        Tipo datos; O=binarios 1=BASIC
IY+#51 =                        Dirección principio datos (octeto
                                inferior)
IY+#52 =                        Dirección principio datos (octeto
                                superior)
IY+#53 =                        Dirección fin datos (octeto inferior)
IY+#54 =                        Dirección fin datos (octeto superior)
IY+#55 a IY+#64 =     Nombre    datos (16 caráct. No usada en
                                CPCNET V1.1)

IY+#65 a IY+#7f =               No usadas en CPCNET V1.1
```

```
                 10 *H+ CPCNET listado de la ROM
                 20 ;
                 30 ;
                 40 ; **         CPCNET     listado de la ROM          **
                 50 ;
                 60 ;
                 70 ; Red de datos de comunicación entre ordenadores
                 80 ; operada manualmente para los ordenadores Amstrad CPC.
                 90 ; Kernow computers (c) 1985
                100 ;
F8F0             110 PTCSND EQU  #F8F0 ; Dirección registros PTC
F8F1             120 PTCREC EQU  #F8F1
F8F2             130 PTCCNT EQU  #F8F2 ; Señales control canales
F8F3             140 PPICTL EQU  #F8F3 ; Registro control PTC chip PPI
0005             150 ROMNUM EQU  5         ; Número zócalo ROM
                 160 ;
                 170 ;
                 180 ;
C000             190        ORG  #C000
                 191 *T+ CPCNET.OBJ
                 200 ;
                 210 ;
C000 01          220        DEFB #1         ; ROM PARALELA
C001 010101      230        DEFB #1,#1,#1   ; Marca 1 Vers 1 rev 1
C004 15C0        240        DEFW NAMTAB     ; Apuntador a tabla de nombres
C006 C33EC0      250        JP   NETGO ; Rutina iniciación en encendido
C009 C382C0      260        JP   NETOPN
C00C C314C2      270        JP   NETCLOS
C00F C36BC2      280        JP   NETSND
C012 C35BC3      290        JP   NETREC
C015 4E455420    300 NAMTAB DEFM "NET INI"  ; Entrada iniciación
C01C D4          310        DEFB "T"+#80
C01D 4E45542E    320        DEFM "NET.OPE"
C024 CE          330        DEFB "N"+#80
C025 4E45542E    340        DEFM "NET.CLOS"
```

```
C02D  C5          350        DEFB "E"+#80
C02E  4E45542E    360        DEFM "NET.SEN"
C035  C4          370        DEFB "D"+#80
C036  4E45542E    380        DEFM "NET.RE"
C03C  C3          390        DEFB "C"+#80
C03D  00          400        DEFB 0          ; Fin tabla nombres
                  410 ;
                  420 ;
                  430 ; ** NETGO Esta rutina inicializa la red de datos a
                  440 ; ** tiempo de encendido. Está limitada a preparar el PPI
                  450 ; ** y crear un área de trabajo en RAM que dejará libre
                  460 ; ** el CPC
C03D  F5          470 NETGO  PUSH AF
C03F  C5          480        PUSH BC
C040  25          490        DEC  H          ; HL contiene el límite superiordel
área
                  500 ;                  de trabajo de 256 octetos
C041  01F3F8      510        LD   BC,PPICTL  ; BC apunta al control del PPI
C044  3EAF        520        LD   A,#AF ; Octeto de modo es AF
C046  ED79        530        OUT  (C),A ; escribirlo
C048  3E00        540        LD   A,0
C04A  FD774E      550        LD   (IY+#4E),A
C04D  FD774F      560        LD   (IY+#4F),A ; limpia octetos indicadores
C050  DDE5        570        PUSH IX
C052  DD215FC0    580        LD   IX,INIMES  ; Decirle al usuario que está
C056  CDF1C4      590        CALL OUTPUT     ; hecho con un mensaje
C059  DDE1        600        POP  IX
C05B  C1          610        POP  BC
C05C  F1          620        POP  AF
C05D  37          630        SCF
C05E  C9          640        RET             ; Restaurar registros, poner acarreo
C05F  0A0D        650 INIMES DEFB #0A,#0D
C061  2A2A2043    660        DEFM "** CPCNET  V1.1 iniciada **   "
C07F  0A0DFF      670        DEFB #0A,#0D,#FF
                  680 ;
                  690 ;
                  700 ; ** NETOPN  Esta rutina del CPCNET abre el PTC como canal
                  710 ; ** 8 del BASIC, permitiendo la trasferencia de versiones
                  720 ; ** en formato texto de programas BASIC sobre el CPCNET,
                  730 ; ** usando sentencias de tipo LIST #8 o PRINT #8
                  740 ;
                  750 ;
                  760 ;
C082  F5          770 NETOPN PUSH AF
C083  E5          780        PUSH HL
C084  FDCB4E46    790        BIT  0,(IY+#4E) ; Ver si bloques salto modificados
C088  203E        800        JR   NZ,MAKHDR  ; saltar esa sección
C08A  FDCB4EC6    810        SET  0,(IY+#4E) ; Si hay que modificarlos, ind. puesto
C08E  FDCB4FC6    820        SET  0,(IY+#4F) ; poner bit actividad
C092  2A2BBD      830        LD   HL,(#BD2B) ; tomar dos octetos
C095  FD7500      840        LD   (IY),L     ; salvar el primero
C098  FD7401      850        LD   (IY+#1),H  ; salvar el segundo
C09B  2A2DBD      860        LD   HL,(#BD2D) ; tomar dos último octetos
C09E  FD7502      870        LD   (IY+#2),L  ; salvar el tercero
C0A1  FD7403      880        LD   (IY+#3),H  ; y el último
C0A4  3EDF        890        LD   A,#DF ; cargar instrucción rearranque
C0A6  322BBD      900        LD   (#BD2B),A  ; en lugar de lo que contenía
C0A9  2164C5      910        LD   HL,SEND    ; HL apunta a rutina SEND
C0AC  FD7510      920        LD   (IY+#10),L ; Poner primer octeto
```

```
C0AF FD7411     930        LD   (IY+#11),H ; Y el segundo
C0B2 2605       940        LD   H,ROMNUM   ; Ahora número ROM
C0B4 FD7412     950        LD   (IY+#12),H ; Número ROM en dirección alta
C0B7 FDE5       960        PUSH IY         ; Copia IY en HL
C0B9 E1         970        POP  HL
C0BA D5         980        PUSH DE         ; salvar DE
C0BB 111000     990        LD   DE,#10
C0BE 19        1000        ADD  HL,DE ; calcula direc. bloque salto
C0BF D1        1010             POP  DE        ; restaurar DE
C0C0 222CBD    1020        LD   (#BD2C),HL ; colocar direc. bloque salto
C0C3 3EC9      1030        LD   A,#C9 ; cargar inst. RET en la entrada
C0C5 322EBD    1040        LD   (#BD2E),A  ; después de PRINT CHAR
C0C8 3E01      1050 MAKHDR LD   A,#1  ; construir mem. int. cabecera
C0CA FD7750    1060        LD   (IY+#50),A ; poner tipo datos (BASIC=01)
C0CD 3E00      1070        LD   A,0        ; cero a direcciones mem. int.
C0CF FD7751    1080        LD   (IY+#51,A)
C0D2 FD7752    1090        LD   (IY+#52,A)
C0D5 FD7753    1100        LD   (IY+#53,A)
C0D8 FD7754    1110        LD   (IY+#54,A) ; cero a campos datos
C0DB FDE5      1120        PUSH IY         ; Copiar IY a HL
C0DD E1        1130        POP  HL
C0DE 015500    1140        LD   BC,#55     ; HL apunta a campo nombre
C0E1 09        1150        ADD  HL,BC
C0E2 0610      1160        LD   B,16  ; 16 caracteres a cero
C0E3 77        1170 OPN009 LD   (HL),A     ; cero un octeto
C0E4 23        1180        INC  HL         ; incrementa apuntador
C0E5 10FC      1190        DJNZ OPN009     ; repite
C0E8 CDC5C4    1200        CALL SNDHDR     ; Mandar cabecera
C0EB DD21F5C0  1210        LD   IX,OPEND   ; Decirselo al usuario
C0EF CDF1C4    1220        CALL OUTPUT
C0F2 C30DC2    1230        JP   OPNEND     ; y terminar
C0F5 0A0D      1240 OPEND  DEFB #0A,#0D
C0F7 2A2A2043  1250        DEFM "** CPCNET esta abierto para mandar programas
                                **"
C127 0A0D      1260        DEFB #0A,#0D
C129 2A2A206F  1270        DEFM "** o variables BASIC por el canal 8. Puede
                                **"
C159 0A0D      1280        DEFB #0A,#0D
C15B 2A2A2075  1290        DEFM "** usar sentencias LIST #8 o PRINT #8 para
                                hacerlo.    **"
C194 0A0D      1300        DEFB #0A,#0D
C196 2A2A2063  1310        DEFM "** cuando haya terminado, ejecute un comando
                                **"
C1CF 0A0D      1320        DEFB #0A,#0D
C1D1 2A2A207C  1330        DEFM "** |NET.CLOSE para terminar la trasmision.
                                **"
C2A0 0A0DFF    1340        DEFB #0A,#0D,#FF
C2AD E1        1350 OPNEND POP  HL
C20E F1        1360        POP  AF         ; restaurar
C20F FDCB4F86  1370        RES  0,(IY+#4F) ; quitar indic. actividad
C213 C9        1380        RET             ; y volver
               1390 ;
               1400 ;
               1410 ; ** NETCLO Esta rutina del CPCNET cierra el canal PTC al
               1420 ; ** BASIC, para entrada y salida. Restaura con sus valores
               1430 ; ** originales las entradas de los bloques de salto que
               1440 ; ** había modificado previamente. Si no hay
               1450 ; ** ninguno modificado, no toma ninguna acción
               1460 ;
```

```
                1470 ;
C214  E5          1480 NETCLO PUSH HL
C215  F5          1490        PUSH AF
C216  FDCB4ECE    1500        SET  1,(IY+#4E) ; Poner bit actividad octeto 2
C21A  FDCB4E46    1510        BIT  0,(IY+#4E) ; entradas de impresión cambiadas?
C21E  2816        1520        JR   Z,CLS001   ; no, saltar
C220  FDCB4E86    1530        RES  0,(IY+#4E) ; limpiar indicador
C224  FD6E00      1540        LD   L,(IY)     ; tomar primer octeto
C227  FD6601      1550        LD   H,(IY+#1)  ; tomar segundo
C22A  222BBD      1560        LD   (#BD2B),HL ; restaurarlos
C22D  FD6E02      1570        LD   L,(IY+#2)  ; tomar tercer octeto
C230  FD6603      1580        LD   H,(IY+#3)  ; tomar último octeto
C233  222DBD      1590        LD   (#BD2D),HL ; restaurarlos
C236  FDCB4F7E    1600 CLS001 BIT  7,(IY+#4F) ; Ver si se han modificado las
C23A  2828        1610        JR   Z,CLSEND   ; entradas CAS xx, no, saltar
C23C  FDCB4FBE    1620        RES  7,(IY+#4F) ; limpiar indicador. Vamos a
C240  FD6E04      1630        LD   L,(IY+#4)  ; restaurarlas. Primer octeto
C243  FD6605      1640        LD   H,(IY+#5)  ; segundo octeto
C246  2277BC      1650        LD   (#BC77),HL ; restaurarlos
C249  FD6E06      1660        LD   L,(IY+#6)  ; tercer octeto
C24C  FD6607      1670        LD   H,(IY+#7)  ; cuarto octeto
C24F  2279BC      1680        LD   (#BC79),HL ; restaurarlos
C252  FD6E08      1690        LD   L,(IY+#8)  ; ahora la entradas de CAS_IN_CHAR
C255  FD6609      1700        LD   H,(IY+#9)  ; restaurarlas
C258  2280BC      1710        LD   (#BC80),HL
C25B  FD6E0A      1720        LD   L,(IY+#A)
C25E  FD660B      1730        LD   H,(IY+#B)
C261  2282BC      1740        LD   (#BC82),HL ; Hecho
C264  F1          1750 CLSEND POP  AF
C265  E1          1760        POP  HL
C266  FDCB4E8E    1770        RES  1,(IY+#4E) ; restaurar bit actividad
C26A  C9          1780        RET             ; y retornar
                  1790 ;
                  1800 ;
                  1810 ; ** NET SEND Se invoca con un comando como -
                  1820 ; ** |NET.SEND,DIR_PRINC,DIR_FIN
                  1830 ; ** La trasmisión se realiza con los octetos comprendidos
                  1840 ; ** entre las direcciones de principio y de fin
                  1850 ;
                  1860 ;
C26B  F5          1870 NETSND PUSH AF
C26C  D5          1880        PUSH DE
C26D  E5          1890        PUSH HL
C26E  DDE5        1900        PUSH IX
C270  C5          1910        PUSH BC
C271  FE02        1920        CP   #2         ; Ver si ha pasado dos parametros
C273  2866        1930        JR   Z,SND001   ; Si, saltar
C275  DD217FC2    1940 SND004 LD   IX,BUMPAR  ; No, msg error
C279  CDF1C4      1950        CALL OUTPUT     ; lanzar msg
C27C  C354C3      1960        JP   SNDEND     ; y adios
C27F  2A2A2044    1970 BUMPAR DEFM "** Debe especificar direcciones   **"
C2A3  0D0A        1980        DEFB #0D,#0A
C2A5  2A2A2064    1990        DEFM "** de principio y final.  Eg-|NET.SEND,1000,2000
                                  **"
C2D8  0A0DFF      2000        DEFB #0A,#0D,#FF
C2DB  FDE5        2010 SND001 PUSH IY         ; Copia IY a HL
C2DD  E1          2020        POP  HL
C2DE  015000      2030        LD   BC,#50
C2E1  09          2040        ADD  HL,BC ; HL apunta mem. int. cabecera
```

```
C2E2  3600      2050        LD   (HL),#00   ; Tipo datos es binario
C2E4  23        2060        INC  HL         ; HL=siguiente octeto
C2E5  DD7E02    2070        LD   A,(IX+2)   ; Comienzo datos (L)
C2E8  77        2080        LD   (HL),A        ; ponerlo en mem. int.
C2E9  DD7E03    2090        LD   A,(IX+3)   ; Comienzo datos (H)
C2EC  23        2100        INC  HL         ; Incrementa apuntador
C2ED  77        2110        LD   (HL),A     ; ponerlo en mem. int.
C2EE  DD7E00    2120        LD   A,(IX)     ; Fin datos (L)
C2F1  23        2130        INC  HL
C2F2  77        2140        LD   (HL),A     ; ponerlo en mem. int.
C2F3  DD7E01    2150        LD   A,(IX+1)   ; fin datos (H)
C2F6  23        2160        INC  HL         ; incrementar apuntador
C2F7  77        2170        LD   (HL),A     ; ponerlo en mem. int.
C2F8  23        2180        INC  HL         ; incrementar apuntador
C2F9  0610      2190        LD   B,16  ; 16 espacios en cabecera
C2FB  3E20      2200        LD   A,#20
C2FD  77        2210 SND002 LD   (HL),A
C2FE  23        2220        INC  HL
C2FF  10FC      2230        DJNZ SND002     ; repetir 16 veces
C301  CDC5C4    2240        CALL SNDHDR     ; mandar cabecera
C304  FD6E51    2250        LD   L,(IY+#51)
C307  FD6652    2260        LD   H,(IY+#52) ; HL apunta princ. datos
C30A  FD4E53    2270        LD   C,(IY+#53) ; y BC a fin datos
C30D  FD4654    2280        LD   B,(IY+#54)
C310  ED42      2290        SBC  HL,BC ; ver si fin > principio
C312  D275C2    2300        JP   NC,SND004  ; si, saltar
C315  FD6E51    2310        LD   L,(IY+#51) ; reacrgar dir. pricipio
C318  FD6652    2320        LD   H,(IY+#52)
C31B  7E        2330 SND003 LD   A,(HL)     ; Tomar un octeto de memoria
C31C  CD64C5    2340        CALL SEND  ; mandarlo
C31F  3833      2350        JR   C,SNDEND   ; Terminar si se pulsó CTRL/A
C321  23        2360        INC  HL         ; incrementar apuntador
C322  7D        2370        LD   A,L        ; copia L en A
C323  FDBE53    2380        CP   (IY+#53)   ; Fin del área (octeto inferior)
C326  20F3      2390        JR   NZ,SND003  ; no, saltar
C328  7C        2400        LD   A,H        ; tomar octeto superior
C329  FDBE54    2410        CP   (IY+#54)   ; ver si coincide
C32C  20ED      2420        JR   NZ,SND003  ; no, saltar
                2430 ; ** Cuando llegamos aquí ya se ha terminado la trasferencia
                2440 ; ** así que manda los mensajes correspondientes
                2450 ;
C32E  DD2137C3  2466        LD   IX,SUCESS
C332  CDF1C4    2470        CALL OUTPUT
C335  181D      2480        JR   SNDEND
C357  0A0D      2490 SUCESS DEFB #0A,#0D
C339  2A2A2054  2500        DEFM "** Transf. completada **"
C351  0A0DFF    2510        DEFB #0A,#0D,#FF
C354  C1        2520 SNDEND POP  BC
C355  DDE1      2530        POP  IX
C357  E1        2540        POP  HL
C358  D1        2550        POP  DE
C359  F1        2560        POP  AF
C35A  C9        2570        RET
                2580 ;
                2590 ;
                2600 ; ** NETREC Esta rutina de CPCNET es el receptor principal
                2610 ; ** de datos desde el canal receptor del PTC. Recibe la
                2620 ; ** cabecera empaquetada y mira si lo que está recibiendo
```

```
                2630 ; ** es un programa BASIC. Si lo es modifica los bloques de
                2640 ; ** salto correspondientes y lanza un mensaje para que
                2650 ; ** introduzca un comando LOAD. Si el tipo de datos
                2660 ; ** recibido es binario, NETREC realiza por sí mismo el
                2670 ; ** trabajo de almacenamiento de los datos recibidos.
                2680 ; ** NOTA cuando recibe datos binarios no comprueba si el
                2690 ; ** área usada para almacenarlos está libre o no.
                2700 ;
                2710 ;
C358  F5        2720 NETREC PUSH AF
C35C  E5        2730        PUSH HL
C35D  C5        2740        PUSH BC
C35E  D5        2750        PUSH DE         ; Usamos todos los registros
C35F  CD8DC5    2760        CALL RECHDR     ; Recibir cabecera
C362  DABCC4    2770        JP   C,RECEND   ; Abortar si CTRL/A
C365  FDCB4FD6  2780        SET  2,(IY+#4F) ; poner bit de actividad
C369  FD7E50    2790        LD   A,(IY+#50) ; obtener octeto de tipo
C36C  FE01      2800        CP   #01        ; ver si es BASIC
C35E  281F      2810        JR   Z,BASREC   ; Si, saltar a receptor BASIC
C370  FD6E51    2820 BINREC LD   L,(IY+#51) ; HL=dirección principio
C373  FD6652    2830        LD   H,(IY+#52)
C376  FD5E53    2840        LD   E,(IY+#53) ; DE=dirección fin
C379  FD5654    2850        LD   D,(IY+#54)
C37C  CD3EC5    2860 REC001 CALL RECEEV     ; Tomar un carácter
C37F  DABCC4    2870        JP   C,RECEND   ; abortar si CTRL/A
C382  7E        2880        LD   A,(HL)     ; almacenar el octeto
C383  7B        2890        LD   A,E        ; ver si el octeto inf. del apunt.
C384  BD        2900        CP   L          ; coincide con el fin
C385  2005      2910        JR   NZ,REC002  ; no, saltar
C387  7A        2920        LD   A,D        ; comparar octetos altos
C388  BC        2930        CP   H          ; ver si coinciden
C389  CABCC4    2940        JP   Z,RECEND   ; si, saltar
C38C  23        2950 REC007 INC  HL         ; incrementar apuntador
C38D  18ED      2960        JR   REC001     ; y repetir
C38F  FDCB4E46  2970 BASREC BIT  0,(IY+#4E) ; ver si está modificado
C393  206F      2980        JR   NZ,REC003  ; si, saltar esta sección
C395  3A77BC    2990        LD   A,(#BC77)  ; cargador BASIC
C398  FD7704    3000        LD   (IY+#4),A  ; salvar primer octeto CAS_IN_OPEN
C39B  2A78BC    3010        LD   HL,(#BC78) ; tomar los otros dos
C39E  FD7505    3020        LD   (IY+#5),L  ; salvarlos en área trabajo
C3A1  FD7406    3030        LD   (IY+#6),H
C3A4  3E05      3040        LD   A,ROMNUM   ; A=número ROM
C3A6  FD7715    3050        LD   (IY+#15),A ; es tercer octeto dir. remota
C3A9  FDE5      3060        PUSH IY
C3AB  E1        3070        POP  HL         ; copiar IY en HL
C3AC  111300    3080        LD   DE,#13     ; HL=dirección remota
C3AF  19        3090        ADD  HL,DE
C3B0  2278BC    3100        LD   (#BC78),HL ; poner bloque de salto
C3B3  3EDF      3110        LD   A,#DF ; y el RST 18
C3B5  3277BC    3120        LD   (#BC77),A  ; también
C3B8  3A7ABC    3130        LD   A,(#BC7A)  ; Salvar el siguiente octeto
C3BB  FD7707    3140        LD   (IY+#7),A  ; Poner un RET
C3BE  3EC9      3150        LD   A,#C9
C3C0  327ABC    3160        LD   (#BC7A),A
C3C3  111FC5    3170        LD   DE,BAS001  ; DE apunta a BAS001
C3C6  73        3180        LD   (HL),E     ; colocarla en dirección remota
C3C7  23        3190        INC  HL
C3C8  72        3200        LD   (HL),D
```

```
C3C9  3A80BC    3210        LD   A,(#BC80)  ; Modificar CAS_IN_CHAR
C3CC  FD7708    3220        LD   (IY+#8),A  ; salvar octetos originales
C3CF  2A81BC    3230        LD   HL,(#BC81)
C3D2  FD7509    3240        LD   (IY+#9),L
C3D5  FD740A    3250        LD   (IY+#A),H
C3D8  3A83BC    3260        LD   A,(#BC83)  ; También el tercer octeto
C3DB  FD770B    3270        LD   (IY+#B),A
C3DE  3EC9      3280        LD   A,#C9 ; poner aquí un RET
C3E0  3283BC    3290        LD   (#BC83),A
C3E3  2125C5    3300        LD   HL,RECBAS
C3E6  FD7516    3310        LD   (IY+#16),H ; Reemplazar la dirección
C3E9  FD7417    3320        LD   (IY+#17),L ; remota de la CAS_IN_CHAR
C3EC  3E05      3330        LD   A,ROMNUN   ; y ROMNUM completa dirección remota
C3EE  FD7718    3340        LD   (IY+#18),A
C3F1  FDE5      3350        PUSH IY
C3F3  E1        3360        POP  HL         ; copiar IY en HL
C3F4  011600    3370        LD   BC,#16     ; BC=despl. a dir. remota
C3F7  09        3380        ADD  HL,BC ; HL=dirección remota
C3F8  3EDF      3390        LD   A,#DF ; RST 18 en bloque salto
C3FA  3280BC    3400        LD   (#BC80),A  ; entrada para CAS_IN_CHAR
C3FD  2281BC    3410        LD   (#BC81),HL ; y después dir. remota
C400  FDCB4FFE  3420        SET  7,(IY+#4F) ; poner bit especial en octeto
                3430 ; función, indica cambio entradas bloque salto E/S cass. C404
DD210EC4        3440 REC003 LD   IX,RECMES  ; Decimos como recibir
C408  CDF1C4    3450        CALL OUTPUT
C40B  C3BCC4    3460        JP   RECEND
C40E  0A0D      3470 RECMES DEFB #0A,#0D
C410  2A2A204C  3480        DEFM "** Los datos que llegan se van a meter en el
                                 BASIC.  **"
C447  0A0D      3490        DEFB #0A,#0D
C449  2A2A2054  3500        DEFM "** Teclee los comandos LOAD o MERGE para empezar
                                 **"
C480  0A0D      3510        DEFB #0A,#0D
C482  2A2A206C  3520        DEFM "** la recepción.                CPCNET
                                 **"
C4B9  0A0DFF    3530        DEFB #0A,#0D,#FF
C4BC  D1        3540 RECEND POP  DE
C4BD  C1        3550        POP  BC
C4BE  E1        3560        POP  HL
C4BF  F1        3570        POP  AF
C4C0  FDCB4F96  3580        RES  2,(IY+#4F) ; Quitar bit de actividad
C4C4  C9        3590        RET             ; y retornar
                3600 ;
                3610 ;
                3620 ; ** SNDHDR Esta subrutina manda el contenido     **
                3630 ; ** DE la memoria intermedia de cabecera al PTC. **
                3640 ; ** IY apunta a la base del área de trabajo      **
                3650 ;
C4C5  F5        3660 SNDHDR PUSH AF
C4C6  E5        3670        PUSH HL
C4C7  C5        3680        PUSH BC
C4C8  FDCB4FE6  3690        SET  4,(IY+#4F) ; Poner bit actividad
C4CC  3EAA      3700        LD   A,#AA
C4CE  CD64C5    3710        CALL SEND  ; mandar octeto marcador
C4D1  3816      3720        JR   C,HDR001   ; saltar si tecla pulsada
C4D3  FDE5      3730        PUSH IY
C4D5  E1        3740        POP  HL         ; Copiar IY en Hl
C4D6  015000    3750        LD   BC,#50
C4D9  09        3760        ADD  HL,BC ; HL apunta a mem. int. cabecera
```

```
C4DA  0630      3770        LD   B,#30 ; B=contador bucle
C4DC  7E        3780 SNDLOP LD   A,(HL)     ; tomar un octeto
C4DD  CD64C5    3890        CALL SEND  ; mandarlo
C4E0  3807      3800        JR   C,HDR001   ; salir si tecla pulsada
C4E2  23        3810        INC  HL         ; incrementar apuntador
C4E3  10F7      3820        DJNZ SNDLOP     ; repetir bucle
C4E5  3E00      3830        LD   A,0        ; limpiar acarreo al final
C4E7  CE00      3840        ADC  0
C4E9  C1        3850 HDR001 POP  BC
C4EA  E1        3860        POP  HL
C4EB  F1        3870        POP  AF
C4EC  FDCB4FA6  3880        RES  4,(IY+#4F) ; Limpiar bit actividad
C4F0  C9        3890        RET             ; retornar
                3900 ;
                3910 ; ** La rutina de salida manda mensajes a la pantalla    **
                3920 ; ** IX apunta al principio del mensaje, que termina con **
                3930 ; ** un octeo de valor #FF. AF, IX y HL corrompidos      **
C4F1  DD7E00    3940 OUTPUT LD   A,(IX)     ; Toma un octeto
C4F4  CB7F      3950        BIT  7,A        ; ver si es EOB
C4F6  C0        3960        RET  NZ         ; si, retornar
C4F7  CD5ABB    3970        CALL #BB5A ; mandar el carácter
C4FA  DD23      3980        INC  IX         ; incrementar apuntador
C4FC  18F3      3490        JR   OUTPUT     ; y repetir
C4FE  67        4000 HEXAS  LD   H,A        ; salvar valor original
C4FF  E60F      4010        AND  #0F        ; dejar mitad inferior
C501  F630      4020        OR   #30        ; convertir en ASCII
C503  FE3A      4030        CP   #3A        ; ver si es alfabético
C505  FA0AC5    4040        JP   M,HIGH     ; no, saltar
C508  C607      4050        ADD  A,#7  ; convertir en hex
C50A  6F        4060 HIGH   LD   L,A        ; Poner dígito inferior en L
C50B  7C        4070        LD   A,H        ; restaurar original
C50C  CB3F      4080        SRL  A          ; poner mitad superior en inferior
C50E  CB3F      4090        SRL  A
C510  CB3F      4100        SRL  A
C512  CB3F      4110        SRL  A
C514  F630      4120        OR   #30
C516  FE3A      4130        CP   #3A        ; ver si es alfabético
C518  FA1DC5    4140        JP   M,DONE     ; no, terminado
C51B  C607      4150        ADD  A,#7  ; ajustar como antes
C51D  67        4160 DONE   LD   H,A        ; salvar en H
C51E  C9        4170        RET             ; y terminar
                4180 ;
                4190 ;
                4200 ; ** BAS001 Esta rutina reemplaza a la rutina CAS_IN_OPEN de
                4210 ; ** la ROM durante la recepción de programas BASIC desde el
                4220 ; ** CPCNET. Lo carga dentro del BASIC como si hubiera
                4230 ; ** abierto un fichero con un programa BASIC en ASCII
                4240 ;
                4250 ;
C51F  3E16      4260 BAS001 LD   A,#16 ; Carga código tipo ASCII
C521  FE00      4270        CP   #0         ; limpia indicador Z
C523  37        4280        SCF             ; pone indicador acarreo
C524  C9        4290        RET             ; vuelve al BASIC
                4300 ;
                4310 ;
                4320 ;
                4330 ;
                4340 ; ** RECBAS Esta rutina reemplaza a la rutina CAS_IN_CHAR de
                4350 ; ** la ROM, que introduce programas BASIC tomados del
```

```
                4360 ; ** cassette o disco, carácter a carácter. Esta rutina usa
                4370 ; ** la subrutina RECEEV e indica el fin de fichero al BASIC
                4380 ; ** si el octeto recibido tiene puesto el bit siete.
                4390 ;
                4400 ;
                4410 ;
C525  CD3EC5    4420 RECBAS CALL RECEEV     ; Toma un carácter
C528  3005      4430        JR   NC,RECB01  ; Salta si no CTRL/A
C52A  3E00      4440        LD   A,0        ; si lo ha tecleado
C52C  CE00      4450        ADC  A,#0  ; pone los bits de cero y acarreo
C52E  C9        4460        RET             ; pasa BREAK al BASIC y retorna
C52F  CB7F      4470 RECB01 BIT  7,A        ; ver si bit 7 puesto
C531  2807      4480        JR   Z,RECB02   ; no, salta
C533  3E00      4490        LD   A,0
C535  CE00      4500        ADC  A,#0
C537  3E1F      4510        LD   A,#1F ; limpia bits cero y acarreo
C539  C9        4520        RET             ; A=EOF y retorna
C53A  FEFF      4530 RECB02 CP   #FF        ; limpiar indicador Z
C53C  37        4540        SCF             ; poner acarreo para decirlo
C53D  C9        4550        RET             ; al BASIC que OK y reotrnar
                4560 ;
                4570 ;
                4580 ; ** RECEEV Esta rutina retorna a otras rutinas los caracte-
                4590 ; ** res que recibe por el canal de trasferencia paralela
                4600 ; ** CPCNET espera a que se ponga en estado lógico alto la
                4610 ; ** señal IBF (mem. int. de entrada llena) de la puerta B,
                4620 ; ** que indica que está disponible el octeto. Mientras
                4630 ; ** espera, llama a la rutina de la ROM que comprueba si se
                4640 ; ** ha tecleado CTRL/A. Si se ha hecho, retorna con el
                4650 ; ** indicador de acarreo puesto. Cuando lee un octeto, lo
                4660 ; ** pasa dentro del registro A.
                4670 ;
                4680 ;
C53E  C5        4690 RECEEV PUSH BC
C53F  FDCB4FEE  4700        SET  5,(IY+#4F) ; Comprobar bit actividad
C543  01F2F8    4710 RECEE2 LD   BC,PTCCNT  ; BC apunta al canal de control
C546  ED78      4720        IN   A,(C) ; Obtener estado control
C548  CB4F      4730        BIT  1,A        ; ver si hay IBF
C54A  200C      4740        JR   NZ,RECEE1  ; si, saltar
C54C  CD09BB    4750        CALL #BB09 ; comprobar CTRL/A
C54F  30F2      4760        JR   NC,RECEE2  ; no, repetir bucle
C551  FE01      4770        CP   #1         ; ver si es CTRL/A
C553  20EE      4780        JR   NZ,RECEE2  ; no, repetir bucle
C555  37        4790        SCF             ; poner acarreo
C556  1806      4800        JR   RECOUT     ; e irse
C558  01F1F8    4810 RECEE1 LD   BC,PTCREC  ; BC apunta a canal recepción
C55B  ED78      4820        IN   A,(C) ; tomas el carácter
C55D  BF        4830        CP   A          ; limpiar acarreo
C55E  C1        4840 RECOUT POP  BC         ; limpiar pila
C55F  FDCB4FAE  4850        RES  5,(IY+#4F) ; quitar bit actividad
C563  C9        4860        RET             ; y volver
                4870 ;
                4880 ;
                4890 ; ** SEND Esta rutina de CPCNET manda al canal de trasmisión
                4900 ; ** del PTC el carácter que le llegaen el registro A, pero
                4910 ; ** primero espera a que el canal está disponibles - indica-
                4920 ; ** do por el bit 7 alto en la puerta C - PTCCNT.
                4930 ; ** Si el usuario pulsa una tecla mientras SEND está espe-
```

```
                4940 ; ** rando, pone el inidicador de acarreo y retorna
                4950 ;
C564  C5          4960 SEND   PUSH BC
C565  E5          4970        PUSH HL
C566  FDCB4FF6    4980        SET  6,(IY+#4F) ; poner bit actividad
C56A  01F2F8      4990 SEND00 LD   BC,PTCCNT  ; BC=registro de estado del PTC
C56D  ED68        5000        IN   L,(C) ; L=estado
C56F  CB7D        5010        BIT  7,L        ; ver canal trasmisión listo
C571  200D        5020        JR   NZ,SEND01  ; si, saltar
C573  F5          5030        PUSH AF         ; salvar el octeto a mandar
C574  CD09BB      5040        CALL #BB09 ; ver si se ha tecleado algo
C577  3004        5050        JR   NC,SEND02  ; no, saltar
C579  F1          5060        POP  AF         ; recuperar AF
C57A  37          5070        SCF             ; poner acarreo
C57B  1809        5080        JR   SEND04     ; y terminar
C57D  F1          5090 SEND02 POP  AF         ; Si no se ha tecleado, recuperar AF
C57E  18EA        5100        JR   SEND00     ; y repetir bucle
C580  01F0F8      5110 SEND01 LD   BC,PTCSND  ; BC apunta a canal trasmisión
C583  ED79        5120        OUT  (C),A ; y manda un octeto
C585  A7          5130        AND  A          ; limpia acarreo
C586  E1          5140 SEND04 POP  HL         ; restaura la pila
C587  C1          5150        POP  BC
C588  FDCB4FB6    5160        RES  6,(IY+#4F) ; limpia bit actividad
C58C  C9          5170        RET             ; retorna
                  5180 ;
                  5190 ;
                  5200 ;
                  5210 ;
                  5220 ; ** RECHDR Esta rutina recive una cabecer del CPCNET desde
                  5230 ; ** el canal de recepción del PTC. La cabecera debe tener el
                  5240 ; ** formato correcto, o mandará un mensaje
                  5250 ;
                  5260 ;
C58D  C5          5270 RECHDR PUSH BC
C58E  E5          5280        PUSH HL
C58F  D5          5290        PUSH DE
C590  FDCB4FDE    5300        SET  3,(IY+#4F) ; poner bit actividad
C594  FDE5        5310        PUSH IY         ; copia IY a HL
C596  E1          5320        POP  HL
C597  015000      5330        LD   BC,#50
C59A  09          5340        ADD  HL,BC ; HL apunta a mem. int. cabecera
C59B  CD3EC5      5350 RECH01 CALL RECEEV     ; toma un octeto
C59E  DA37C6      5360        JP   C,RECH04   ; salir si lo quiere el usuario
C5A1  FEAA        5370        CP   #AA        ; ver si es marcador principio
C5A3  28F6        5380        JR   Z,RECH01   ; no, repetir bucle
C5A5  CD3EC5      5390        CALL RECEEV     ; si, comienza recepción
C5A8  5F          5300        LD   E,A        ; validar tipo datos (solo 0 o 1)
C5A9  E6FE        5410        AND  #FE        ; enmascarar bit ilegales
C5AB  287D        5420        JR   Z,RECH02   ; si no hay bits puestos, saltar
C5AD  DD21B7C5    5430        LD   IX,RECHER  ; los hay, lanzar mensaje
C5B1  CDF1C4      5440        CALL OUTPUT     .
C5B4  C337C6      5450        JP   RECH04     ; y abortar recepción
C5B7  0A0D        5460 RECHER DEFB #0A,#0D
C5B9  2A2A204D    5470        DEFM "** Atencion: RECHDR detecto el tipo de datos
                                   ilegal      **"
C5EF  0A0D        5480        DEFB #0A,#0D
C5F1  2A2A2072    5490        DEFM "** recepcion cabecera abortada - avisar al
                                  emisor **"
C627  0A0DFF      5500        DEFB #0A,#0D,#FF
```

```
C62A  73        5510 RECH02 LD   (HL),E     ; tipo datos en mem. int.
C62B  23        5520        INC  HL         ; incrementar apuntador
C62C  061B      5530        LD   B,#1B ; 20 octetos más para cabecera
C62E  CD3EC5    5540 RECH03 CALL RECEEV     ; tomar un octeto
C631  3804      5550        JR   C,RECH04   ; abortar si lo quiere el usuario
C633  77        5560        LD   (HL),A     ; almacenar octeto
C634  23        5570        INC  HL         ; actualizar apuntador
C635  10F7      5580        DJNZ RECH03     ; repetir hasta terminar
C637  D1        5590 RECH04 POP  DE
C638  E1        5600        POP  HL
C639  C1        5610        POP  BC         ; restaurar registros
C63A  FDCB4F9E  5620        RES  3,(IY+#4F) ; limpiar bit actividad
C63E  C9        5630        RET             ; retornar

Pass 2 errors: 00

BAS001 C51F  BASREC C38F  BINREC C370  BUMPAR C27F
CLS001 C236  CLSEND C264  DONE   C51D  HDR001 C4E9
HEXAS  C4FE  HIGH   C50A  INIMES C05F  MAKHDR C0C8
NAMTAB C015  NETCLO C214  NETGO  C03E  NETOPN C082
NETREC C35B  NETSND C26B  OPEND  C0F5  OPN009 C0E4
OPNEND C20D  OUTPUT C4F1  PPICTL F8F3  PTCCNT F8F2
PTCREC F8F1  PTCSND F8F0  REC001 C37C  REC002 C38C
REC003 C404  RECB01 C52F  RECB02 C53A  RECBAS C525
RECEE1 C558  RECEE2 C543  RECEEV C53E  RECEND C4BC
RECH01 C59B  RECH02 C62A  RECH03 C62E  RECH04 C637
RECHDR C58D  RECHER C5B7  RECMES C40E  RECOUT C55E
ROMNUM 0005  SEND   C564  SEND00 C56A  SEND01 C580
SEND02 C57D  SEND04 C586  SND001 C2DB  SND002 C2FD
SND003 C31B  SND004 C275  SNDEND C354  SNDHDR C4C5
SNDLOP C4DC  SUCESS C337

Table used:   758  from  2415
```

Fig 6.8

En la figura 6.9 le proporcionamos el vuelco hexadecimal de la ROM del CPCNET.

```
01 01 01 01 15 C0 C3 3E C0 C3 82 C0 C3 14 C2 C3
6B C2 C3 5B C3 4E 45 54 20 49 4E 49 D4 4E 45 54
2E 4F 50 45 CE 4E 45 54 2E 43 4C 4F 53 C5 4E 45
54 2E 53 45 4E C4 4E 45 54 2E 52 45 C3 00 F5 C5
25 01 F3 F8 3E AF ED 79 3E 00 FD 77 4E FD 77 4F
DD E5 DD 21 5F C0 CD F1 C4 DD E1 C1 F1 37 C9 0A
0D 2A 2A 20 43 50 43 4E 45 54 20 20 56 31 2E 31
20 69 6E 69 63 69 61 64 61 20 20 20 20 2A 2A 0A
0D FF F5 E5 FD CB 4E 46 20 3E FD CB 4E C6 FD CB
4F C6 2A 2B BD FD 75 00 FD 74 01 2A 2D BD FD 75
02 FD 74 03 3E DF 32 2B BD 21 64 C5 FD 75 10 FD
74 11 26 05 FD 74 12 FD E5 E1 D5 11 10 00 19 D1
22 2C BD 3E C9 32 2E BD 3E 01 FD 77 50 3E 00 FD
77 51 FD 77 52 FD 77 53 FD 77 54 FD E5 E1 01 55
00 09 06 10 77 23 10 FC CD C5 C4 DD 21 F5 C0 CD
F1 C4 C3 0D C2 0A 0D 2A 2A 20 43 50 43 4E 45 54
```

```
20 65 73 74 61 20 61 62 69 65 72 74 6F 20 70 61
72 61 20 6D 61 6E 64 61 72 20 70 72 6F 67 72 61
6D 61 73 20 20 2A 2A 0A 0D 2A 2A 20 6F 20 76 61
72 69 61 62 6C 65 73 20 42 41 53 49 43 20 70 6F
72 20 65 6C 20 63 61 6E 61 6C 20 38 2E 20 50 75
65 64 65 20 20 20 20 2A 2A 0A 0D 2A 2A 20 75 73
61 72 20 73 65 6E 74 65 6E 63 69 61 73 20 4C 49
53 54 20 23 38 20 6F 20 50 52 49 4E 54 20 23 38
20 70 61 72 61 20 68 61 63 61 72 6C 6F 2E 20 20
20 20 2A 2A 0A 0D 2A 2A 20 63 75 61 6E 64 6F 20
68 61 79 61 20 74 65 71 6D 69 6E 61 64 6F 2C 20
65 6A 65 63 75 74 65 20 75 6E 20 63 6F 6D 61 6E
64 6F 20 20 20 20 20 20 20 20 20 20 20 20 2A 2A 0A
0D 2A 2A 20 7C 4E 45 54 2E 43 4C 4F 53 45 20 70
61 72 61 20 74 65 72 6D 69 6E 61 72 20 6C 61 20
74 72 61 73 6D 69 73 69 6F 6E 2E 20 20 20 20 20
20 20 20 20 20 20 20 20 2A 2A 0A 0D FF E1 F1 FD
CB 4F 86 C9 E5 F5 FD CB 4E CE FD CB 4E 46 28 16
FD CB 4E 86 FD 6E 00 FD 66 01 22 2B BD FD 6E 02
FD 66 03 22 2D BD FD CB 4F 7E 28 28 FD CB 4F BE
FD 6E 04 FD 66 05 22 77 BC FD 6E 06 FD 66 07 22
79 BC FD 6E 08 FD 66 09 22 80 BC FD 6E 0A FD 66
0B 22 82 BC F1 E1 FD CB 4E 8E C9 F5 D5 E5 DD E5
C5 FE 02 28 66 DD 21 7F C2 CD F1 C4 C3 54 C3 2A
2A 20 44 65 62 65 20 65 73 70 65 63 66 66 69 63
61 72 20 64 69 72 65 63 63 69 6F 6E 65 73 20 20
20 2A 2A 0D 0A 2A 2A 20 64 65 20 70 72 69 6E 63
69 70 69 6F 20 79 20 66 66 6E 61 6C 2E 20 20 45
6A 2D 7C 4E 45 54 2E 53 45 4E 44 2C 31 30 30 30
2C 32 30 30 30 20 2A 2A 0A 0D FF FD E5 E1 01 50
00 09 36 00 23 DD 7E 02 77 DD 7E 03 23 77 DD 7E
00 23 77 DD 7E 01 23 77 23 06 10 3E 20 77 23 10
FC CD C5 C4 FD 6E 51 FD 66 52 FD 4E 53 FD 46 54
ED 42 D2 75 C2 FD 6E 51 FD 66 52 7E CD 64 C5 38
33 23 7D FD BE 53 20 F3 7C FD BE 54 20 ED DD 21
37 C3 CD F1 C4 18 1D 0A 0D 2A 2A 20 54 72 61 6E
73 66 2E 20 63 6F 6D 70 6C 65 74 61 64 61 20 2A
2A 0A 0D FF C1 DD E1 E1 D1 F1 C9 F5 E5 C5 D5 CD
8D C5 DA BC C4 FD CB 4F D6 FD 7E 50 FE 01 28 1F
FD 6E 52 FD 66 53 FD 5E 53 FD 56 54 CD 3E C5 DA
BC C4 7E 7B BD 20 05 7A BC CA BC C4 23 18 ED FD
CB 4E 46 20 6F 3A 77 BC FD 77 04 2A 78 BC FD 75
05 FD 74 06 3E 05 FD 77 15 FD E5 E1 11 13 00 19
22 78 BC 3E DF 32 77 BC 3A 7A BC FD 77 07 3E C9
32 7A BC 11 1F C5 73 23 72 3A 80 BC FD 77 08 2A
81 BC FD 75 09 FD 74 0A 3A 83 BC FD 77 0B 3E C9
32 83 BC 21 25 C5 FD 75 16 FD 74 17 3E 05 FD 77
18 FD E5 E1 01 16 00 09 3E DF 32 80 BC 22 81 BC
FD CB 4F FE DD 21 0E C4 CD F1 C4 C3 BC C4 0A 0D
2A 2A 20 4C 6F 73 20 64 61 74 6F 73 20 71 75 65
20 6C 6C 65 67 61 6E 20 73 65 20 76 61 6E 20 61
20 6D 65 74 65 72 20 65 6E 20 65 6C 20 42 41 53
49 43 2E 20 20 2A 2A 0A 0D 2A 2A 20 54 65 63 6C
65 65 20 6C 6F 73 20 63 6F 6D 61 6E 64 6F 73 20
4C 4F 41 44 20 6F 20 4D 45 52 47 45 20 70 61 62
61 20 65 6D 70 65 7A 61 62 20 20 20 20 20 2A 2A
0A 0D 2A 2A 20 6C 61 20 62 65 63 65 70 63 69 6F
6E 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 43 50 43 4E 45 54 20 20 20 20 20 20 20
```

```
20 20 20 20 20 20 20 2A 2A 0A 0D FF D1 C1 E1 F1
FD CB 4F 96 C9 F5 E5 C5 FD CB 4F E6 3E AA CD 64
C5 38 16 FD E5 E1 01 50 00 09 06 30 7E CD 64 C5
38 07 23 10 F7 3E 00 CE 00 C1 E1 F1 FD CB 4F A6
C9 DD 7E 00 CB 7F C0 CD 5A BB DD 23 18 F3 67 E6
0F F6 30 FE 3A FA 0A C5 C6 07 6F 7C CB 3F CB 3F
CB 3F CB 3F F6 30 FE 3A FA 1D C5 C6 07 67 C9 3E
16 FE 00 37 C9 CD 3E C5 30 05 3E 00 CE 00 C9 CB
7F 28 07 3E 00 CE 00 3E 1F C9 FE FF 37 C9 C5 FD
CB 4F EE 01 F2 F8 ED 78 CB 4F 20 0C CD 09 BB 30
F2 FE 01 20 EE 37 18 06 01 F1 F8 ED 78 BF C1 FD
CB 4F AE C9 C5 E5 FD CB 4F F6 01 F2 F8 ED 68 CB
7D 20 0D F5 CD 09 BB 30 04 F1 37 18 09 F1 18 EA
01 F0 F8 ED 79 A7 E1 C1 FD CB 4F B6 C9 C5 E5 D5
FD CB 4F DE FD E5 E1 01 50 00 09 CD 3E C5 DA 37
C6 FE AA 28 F6 CD 3E C5 5F E6 FE 28 7D DD 21 B7
C5 CD F1 C4 C3 37 C6 0A 0D 2A 2A 20 41 74 65 6E
63 69 6F 6E 3A 20 52 45 43 48 44 52 20 64 65 74
65 63 74 6F 20 65 6C 20 74 69 70 6F 20 64 65 20
64 61 74 6F 73 20 69 6C 65 67 61 6C 20 2A 2A 0A
0D 2A 2A 20 72 65 63 65 70 63 69 6F 6E 20 63 61
62 65 63 65 72 61 20 61 62 6F 72 74 61 64 61 20
2D 20 61 76 69 73 61 72 20 61 6C 20 65 6D 69 73
6F 72 20 20 20 2A 2A 0A 0D FF 73 23 06 1B CD 3E
C5 38 04 77 23 10 F7 D1 E1 C1 FD CB 4F 9E C9
```

Fig 6.9

CONCLUSIÓN DEL CPCNET

El CPCNET, tal como lo hemos presentado aquí, tiene muchas posibilidades de expansión, y espero que los lectores lo encuentren fácil de expandir y adaptar a sus necesidades. Si alguna vez se produce una segunda versión, espero poder añadir algunas facilidades más, como pantalla interactiva de los datos recibidos, por ejemplo. Me agradaría saber que algún lector ha desarrollado algún nuevo uso para el CPCNET, aunque no creo que pudiera responderle.

CONCLUSIONES DEL CAPÍTULO SEIS

Al final tuvimos que incluir el CPCNET en un capítulo propio. Aunque es otro proyecto más, su complejidad ha hecho que extendamos de esta manera el libro. Las máquinas CPC son muy potentes, y debido al diseño de su sistema operativo y su buena documentación, son una invitación a la investigación y desarrollo de aplicaciones complejas. Espero encontrar muchas más para publicaciones futuras.

APENDICE UNO
Símbolos de Circuitos y Notas de Construcción

Recomendamos encarecidamente a los lectores con poca o ninguna experiencia en 'hardware' de ordenadores, que lean el apéndice seis, titulado "Empiece aquí". Esto les dará unos conocimientos que le servirán para entender mejor el resto del libro. Ya que este no es un libro dedicado específicamente a los principiantes, el apéndice seis es adecuado como introducción general, pero le recomendamos que se suscriba a una de las publicaciones regulares que hay sobre electrónica, para llenar las lagunas que se le puedan presentar.

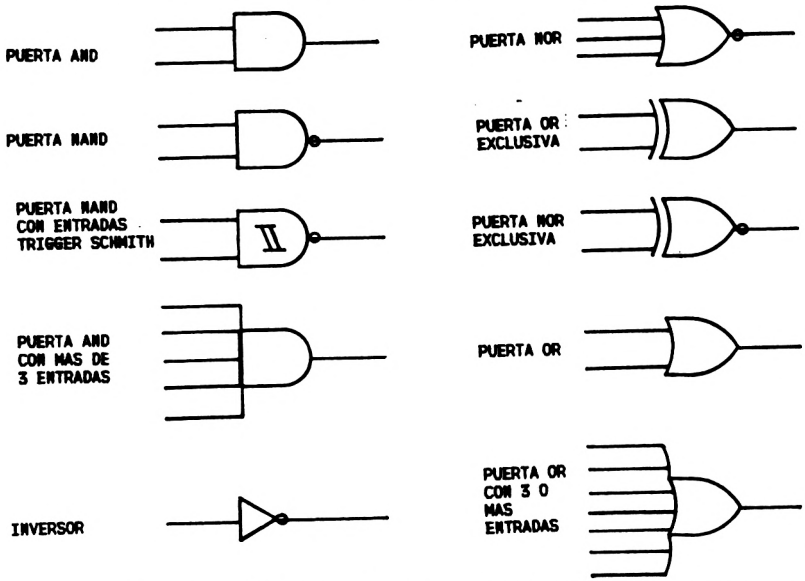
Los esquemas de este libro se han dibujado usando símbolos de elementos lógicos que se adaptan a las normas de American Mil spec. La figura AP1.1 muestra estos símbolos y su significado. Hay muchos otros símbolos que puede encontrar el lector en las revistas, pero los de Mil spec son los más extendidos. Una ventaja adicional es que la documentación técnica de Amstrad usa también estos símbolos - por lo que se mantiene la compatibilidad de la documentación.

En cada proyecto 'hardware' de este libro encontrará lo siguiente:

1) El esquema o esquemas del circuito.
2) Una descripción exhaustiva del funcionamiento del circuito, cómo usarlo y los extras disponibles.
3) El listado de los componentes del circuito.
4) Listado de los programas para hacer funcionar el proyecto.

Los esquemas se ha comprobado cuidadosamente, y están totalmente libres de errores. Las descripciones están diseñadas para darle un conocimiento completo de cómo funciona el proyecto. Las listas de componentes especifican todos los componentes necesarios. Si no encuentra alguno de los componentes podrá mirar en los catálogos de los comercios para encontrar el equivalente, en esto le puede ayudar el personal de la tienda donde los compre. De todas formas tenga cuidado si usa equivalentes, especialmente si no tiene mucha experiencia. Si tiene alguna duda pregunte a alguien que le inspire total confianza. Queremos mejorar su ordenador, no quemarlo. La lista de las conexiones de alimentación de los chips usados, que incluye cada proyecto, incluye también una breve descripción funcional del chip. Esto le dará una referencia para cuando lea el esquema de ese proyecto. Por último, el 'software'. Donde ha sido posible, he incluido un programa BASIC para manejarlo. Esto se debe a que no he querido excluir a los lectores que no poseen un ensamblador como el DEVPAC. Sin embargo hay varios proyectos donde, por razones de velocidad y facilidad, no es posible usar el BASIC. En estos casos se han desarrollado los programas en código máquina. Sin embargo no se ha perdido todo para los lectores que solo conocen el BASIC, en el capítulo cinco encontrará la forma de convertir en listas de códigos BASIC los pequeños progra-

Fig AP1.1 Símbolos lógicos usados en este libro



mas en código máquina. En todos los casos se ha comprobado el 'software' lo más exhaustivamente posible, y se han corregido todos los errores. Si encuentra algún error, por favor háganoslo saber a través del editor, cuya dirección encontrará en las primeras páginas de este libro.

El esquema general para la conexión del CPC gira alrededor del uso de una tarjeta maestra de expansión del 'bus', que ha sido creado por Halstead Designs especialmente para este libro. La figura AP5.2 muestra como se conecta la tarjeta maestra de expansión en el esquema general. Las conexiones del 'bus' de expansión están disponibles al otro extremo de la tarjeta maestra, de forma que puede conectar su unidad de disco o cualquier otro periférico. En el apéndice cinco se explican varios detalles del 'bus' de expansión y otros detalles de conexión.

En cuanto a las fuentes de alimentación, Amstrad me informó que la línea de +5V que se extiende en el conector del 'bus' de expansión, tiene solo 100 miliamperios de capacidad extra. Por esta razón , y por el hecho de que se requieren otras tensiones, además de los +5V, en el proyecto del interface RS232, se necesita una fuente de alimentación extra. (Al principio del capítulo tres se dan detalles de la fuente de alimentación necesaria).

El apéndice tres contiene reproducciones de las hojas de características de los fabricantes tanto de los principales chips usados en el CPC, como de los usados en los proyectos presentados en este libro.

APENDICE DOS
Bibliografía

La siguiente es una pequeña lista de los libros que tratan de los temas tratados en este libro. Las opiniones expresadas sobre ellos son mías propias.

1. **Programming the Z80** por Rodnay Zaks. Publicado en 1982 por Sybex.

Este libro es referido ampliamente como el trabajo más importante sobre el microprocesador Z80. He usado las palabras "más importante" porque contiene una excelente descripción del juego de instrucciones. El libro contiene también una gran cantidad de subrutinas y muestra técnicas de programación y de E/S. A lo largo del libro hay ejercicios para comprobar su comprensión de los puntos explicados, y tiene también gran cantidad de descripciones de los chips que soporta el Z80. Yo prefiero, personalmente, el libro de Barden, pero debido a la excelente sección del juego de instrucciones, le recomiendo que se haga con el libro de Zaks.

2. **The Z80 microcomputer handbook** por William Barden, Jr. Publicado en 1980 por Sams publications.

Este es mi libro favorito sobre Z80. Se introduce más en profundidad en los tiempos de las señales, y en temas generales de 'hardware', que el libro de Zaks. Por ejemplo, describe en detalle el interface para un controlador de disco y da muy buena información sobre interconexión. Es un libro sencillo, que hace las cosas fáciles de entender y encontrar. El juego de instrucciones se explica adecuadamente - aunque no demasiado. Lea primero este libro, y después el de Zaks.

3. **Build your own Z80 computer** por Steve Ciarcia. Publicado en 1981 por Byte books.

Ciarcia escribe aparentemente para la revista Byte (o lo hizo en 1981), y pienso que este libro refleja su experiencia en el campo de las revistas. El libro le lleva paso a paso a través del diseño de un ordenador con una CPU Z80. Hay una descripción del Z80 y de su juego de instrucciones, después la descripción de cómo conectar la memoria, un teclado hexadecimal, y de hacer varios dispositivos periféricos. La máquina terminada está bastante por debajo del nivel del CPC, pero la experiencia de hacerla y entenderla le dará un nivel de conocimientos que le dejará preparado para cualquier otro proyecto de 'hardware' o sistema que usted quiera construir.

Por otro lado, los chips usados son bastante antiguos (aunque todavía se pueden encontrar equipos comerciales que los usan), pero no obstante funcionan, y se pueden encontrar bastante baratos. La sección del diseño de la fuente de alimentación es bastante buena, y es muy importante el listado del ensamblaje de un pequeño programa monitor, para permitirle usar el teclado hexadecimal y el pequeño ordenador para programar en código máquina. Tiene muchos apéndices con

información general.

4. **Guia avanzada del Amstrad CPC 464** por Mark Harrison. Publicada en español por editorial RA-MA en 1985.

Este volumen es uno de los que le recomiendo, ya que cubre toda la parte de 'software' del CPC 464. Encontrará en él datos sobre BASIC, estructuras de datos, codificación binaria y mapa de memoria del CPC 464, por lo que es altamente recomendado y de incalculable valor.

5. **SOFT 158. The CPC firmware specification.** Publicado por Amstrad.

Si usted quiere hacer algo más interesante que cargar juegos en su CPC, debe conseguir este libro. Le explica el funcionamiento de la máquina y tambièn sus 'secretos' internos. Tiene cientos de rutinas detalladas, con descripción de sus condiciones de entrada y salida. ¡CONSIGA ESTE LIBRO!

Si le he contagiado algo de mi pasión por la electrónica mientras ha leído este libro, sería interesante para usted que se mantuviera al día de los nuevos dispositivos, con idea de irlos incorporando en sus propios diseños. La mejor forma de hacerlo es consiguiendo los libros de datos de los fabricantes. Otra forma de acumular información sobre los chips es suscribirse a una revista de electrónica, como Practical Electronics, ETI, Hobby electronics, Wireless and Electronic world, Byte o las revistas en español Elektor, Nueva electrónica, etc. Por último, una de las formas más baratas de obtener información de los chips es en los catálogos de los establecimientos especializados - todos ellos contienen datos útiles de los chips y de un amplio rango de componentes aplicables al diseño de ordenadores.

APENDICE TRES
Datos útiles de los chips

SP0256

Narrator™ Speech Processor

- Natural Speech
- Stand Alone Operation with Inexpensive Support Components
- Wide Operating Voltage
- Word, Phrase, or Sentence Library, ROM Expandable
- Expandable to 491K of ROM Directly
- Simple Interface to Most Microcomputers or Microprocessors
- Supports L.P.C. Synthesis: Formant Synthesis: Allophone Synthesis

GENERAL DESCRIPTION

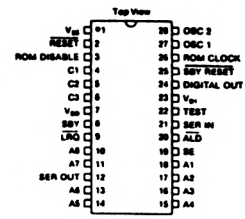
The SP0256 (Speech Processor) is a single chip N-Channel MOS LSI device that is able, using its stored program, to synthesize speech or complex sounds.

The achievable output is equivalent to a flat frequency response ranging from 0 to 5KHz, a dynamic range of 42dB, and a signal to noise ratio of approximately 35dB.

The SP0256 incorporates four basic functions:

- A software programmable digital filter that can be made to model a VOCAL TRACT.
- A 16K ROM which stores both data and instructions (THE PROGRAM).
- A MICROCONTROLLER which controls the data flow from the ROM to the digital filter, the assembly of the "word strings" necessary for linking speech elements together, and the amplitude and pitch information to excite the digital filter.
- A PULSE WIDTH MODULATOR that creates a digital output which is converted to an analog signal when filtered by an external low pass filter.

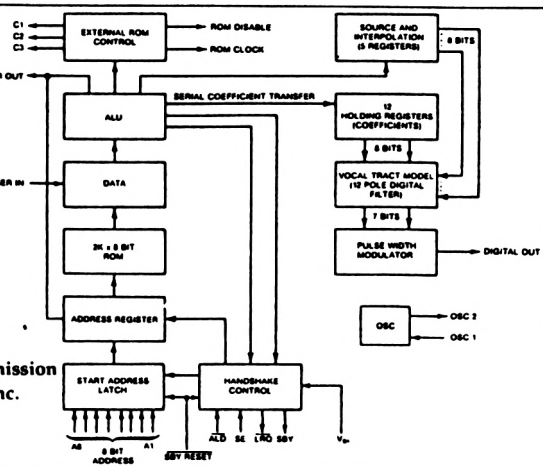
PIN CONFIGURATION
28 LEAD DUAL IN LINE



APPLICATIONS

- Telecommunications
- Appliances
- Computer Peripherals
- Automotive
- Personal Computers
- Toys/Games
- Educational Aids
- Warning Systems
- Security Systems
- Electronic Musical Instruments
- Aids to the Blind
- Narrow Bandwidth
- Communication Systems

SP0256 BLOCK DIAGRAM



SP0256

PROCESADOR DE PALABRAS NARRATOR (TM)

Voz Natural
Operación Independiente sin componentes caros de soporte
Tensión amplia de operación
Librería de Palabras, Frases o Sentencias. Expandible con ROM.
Expandible directamente a 491K de ROM
Interconexión simple con la mayoría de los microordenadores y Microprocesadores
Soprta síntesis de LPC, de Fonemas y Alófonos.

Descripción General

El SP0256 (Procesador de palabras) es un dispositivo de un solo chip MOS N-canal LSI, que es capaz, usando su programa interno, de sintetizar palabras y sonidos complejos.

La salida disponible es equivalente a una primera frecuencia de respuesta en un rango de 0 a 5KHz, un rango dinámico de 42dB, y una media señal/ruido de unos 35dB.

El SP0256 incorpora cuatro funciones básicas:

- Un filtro digital programable que puede imitar al TRACTO BUCAL.
- Una ROM de 16K que almacena los datos y las instrucciones (EL PROGRAMA)
- Un MICROCNTROLADOR que controla el flujo de datos desde la ROM al filtro digital, el ensamblaje de "cadenas de palabras" necesarias para conectar los elementos de la frase, y la información de amplitud y tono para excitar el filtro digital.
- Un MODULADOR DE ANCHO DE IMPULSOS que crea una salida digital que es convertida en analógica cuando es filtrada por un filtro externo de paso bajo.

APLICACIONES

Telecomunicaciones
Aplicaciones
Periféricos de ordenador
Automoción
Ordenadores personales
Juegos/juguetes
Ayuda a la educación
Sistemas de alarma
Sistemas de seguridad
Instrumentos musicales electrónicos
Ayuda a los ciegos
Banda ancha
Sistemas de comunicaciones

Traducción de la hoja de características del chip SP0256

Gothic Crellon Limited

380 Bath Road Slough Berks SL1 6JE England
Telephone Burnham (06286) 4300 telex 847571

Sales telephone Burnham (06286) 4434 telex 847571
Birmingham 021 643 6385 telex 339731

MC6845

Advance Information

MOS

(N-Channel, Silicon-Gate)

CRT CONTROLLER (CRTC)

CRT CONTROLLER (CRTC)

The MC6845 CRT Controller performs the interface to raster scan CRT displays. It is intended for use in processor-based controllers for CRT terminals in stand-alone or cluster configurations.

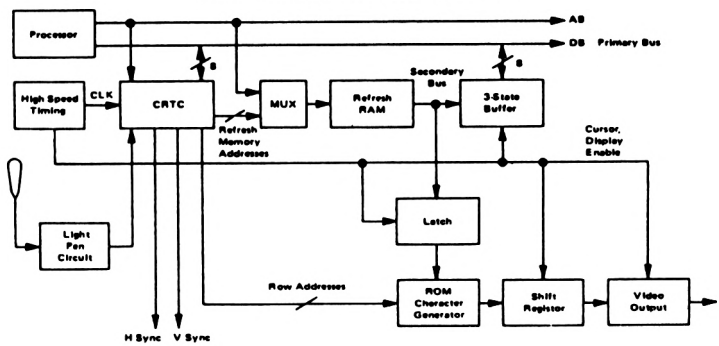
The CRTC is optimized for hardware/software balance in order to achieve integration of all key functions and maintain flexibility. For instance, all keyboard functions, R/W, cursor movements, and editing are under processor control; whereas the CRTC provides video timing and Refresh Memory Addressing.

- Applications include "glass-teletype," smart, programmable, intelligent CRT terminals; video games; information display.
- Alphanumeric, semi-graphic, and full graphic capability.
- Fully programmable via processor data bus. Can generate timing for almost any alphanumeric screen density, e.g. 80 x 24, 72 x 64, 132 x 20, etc.
- Single +5 volt supply. TTL/6800 compatible I/O.
- Hardware scroll (paging or by line or by character)
- Compatible with CPU's and MPU's which provide a means for synchronizing external devices.
- Cursor register and compare circuitry.
- Cursor format and blink are programmable.
- Light pen register.
- Line buffer-less operation. No external DMA required. Refresh Memory is multiplexed between CRTC and MPU.
- Programmable interlace or non-interlace scan.
- 14-bit wide refresh address.

L SUFFIX
CERAMIC PACKAGE
CASE 715

NOT SHOWN:
P SUFFIX
PLASTIC PACKAGE
CASE 711

FIGURE 1 – TYPICAL CRT CONTROLLER APPLICATION



MC6845

CONTROLADOR DE CRT (CRTC)

CONTROLADOR CRT (TUBO DE RAYOS CATODICOS)

El Controlador de CRT MC6845 realiza el interfase con las pantallas CRT. Está diseñado para ser usado en controladores basados en procesadores para terminales CRT aislados o agrupados.

El CRTC está optimizado para un balance 'hardware/software' para conseguir la integración de todas las funciones clave y mantener la flexibilidad. Por ejemplo, todas las funciones del teclado, Lectura/Escritura, movimientos del cursor, y edición están bajo control del procesador; mientras el CRTC proporciona temporización de vídeo y direccionamiento de refresco de memoria.

- Sus aplicaciones incluyen terminales de teletypo elegantes, programables, e inteligentes; vídeo juegos; pantallas de información.
- Capacidad alfa-numérica, semi-gráfica y gráfica completa.
- Via de 'bus' totalmente programable desde el procesador. Puede generar temporizaciónes para casi todas las densidades de pantalla alfa-numéricas, ej. 80 x 24, 72 x 64, 132 x 20, etc.
- Alimentado con solo +5V. E/S compatible TTL/6800
- Movimiento de pantalla por 'hardware' (por páginas o por carácter)
- Compatible con MPUs y CPUs que proporcionan una forma de sincronización de los dispositivos externos.
- Circuitos de registro de cursor y de comparación.
- Forma y parpadeo del cursor, programables.
- Registro para lápiz luminoso.
- Operacion sin memoria intermedia de línea. No requiere DMA externo. El refresco de memoria es multiplexado entre el CTRC y la MPU.
- Análisis programable de interfase o no interfase.
- Dirección de refresco de 14 bits de ancho.

Traducción de la hoja de características del chip MC6845

intel

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40-Pin Dual In-Line Package
- Reduces System Package Count
- Improved DC Driving Capability

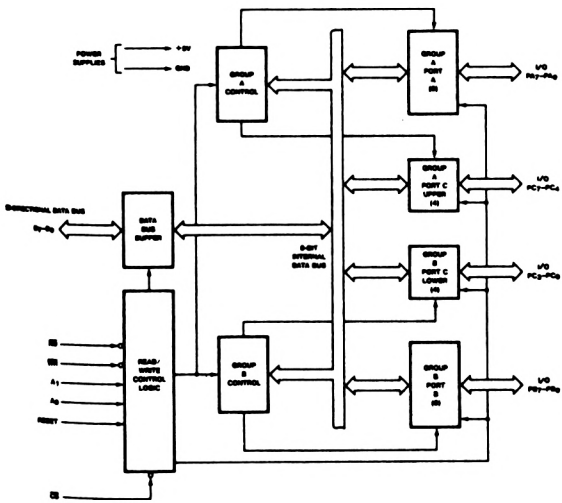


Figure 1. 8255A Block Diagram

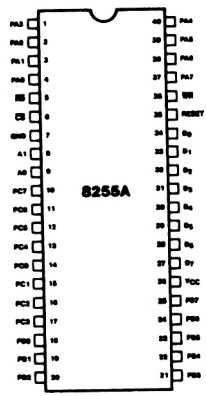


Figure 2. Pin Configuration

8255A/8255A-5

INTERFASE DE PERIFERICOS PROGRAMABLE

- 8255A-5 compatible con MCS-85 (TM).
- 24 Patillas de E/S programables.
- Completamente compatible TTL.
- Totalmente compatible con las familias de Microprocesadores de Intel.
- Características de temporización mejoradas.
- Capacidad de poner/quitar un bit directamente, facilitando el interfase con aplicaciones de control.
- Empaquetado en DIL de 40 patas.
- Reduce el costo de paquetes de sistemas.
- Capacidad de manejo DC mejorada.

Traducción de la hoja de características del chip 8255A

Z8400
Z80® CPU Central Processing Unit

Zilog

Product Specification

September 1983

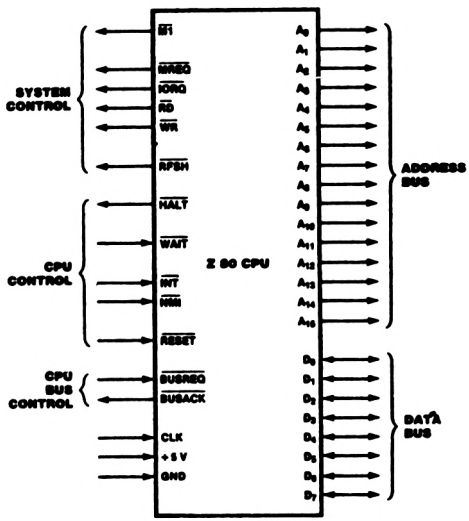


Figure 1. Pin Functions

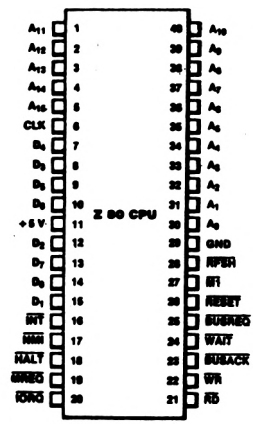


Figure 2. Pin Assignments

8251A
PROGRAMMABLE COMMUNICATION INTERFACE

- **Synchronous and Asynchronous Operation**
- **Synchronous 5-8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion**
- **Asynchronous 5-8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling**
- **Synchronous Baud Rate—DC to 64K Baud**
- **Asynchronous Baud Rate—DC to 19.2K Baud**
- **Full-Duplex, Double-Buffered Transmitter and Receiver**
- **Error Detection—Parity, Overrun and Framing**
- **Compatible with an Extended Range of Intel Microprocessors**
- **28-Pin DIP Package**
- **All Inputs and Outputs are TTL Compatible**
- **Single +5V Supply**
- **Single TTL Clock**

The Intel® 8251A is the enhanced version of the industry standard, Intel 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel's microprocessor families such as MCS-48, 80, 85, and iAPX-86, 88. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM "bi-sync"). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPTY. The chip is fabricated using N-channel silicon gate technology.

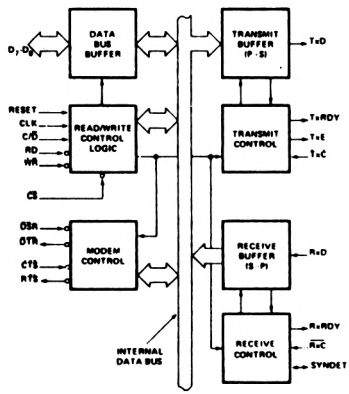


Figure 1. Block Diagram

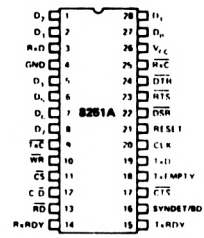


Figure 2. Pin Configuration

MAY 1981
AFN-01573C

8251A

INTERFASE PROGRAMABLE DE COMUNICACIONES

- Operación Síncrona y Asíncrona.
- Caracteres síncronos de 5 a 8 bits; Sincronización de carácter interna o externa; inserción síncrona automática.
- Caractéres asíncronos de 5 a 8 bits; Frecuencia de reloj de 1, 16 o 64 veces la velocidad de transmisión; generación de carácter de ruptura; 1, 1/2 o 2 bits de parada; falsa detección de bit de comienzo; detección y manejo automático de ruptura.
- Velocidad de transmisión síncrona - DC a 64K Baudios.
- Velocidad de transmisión asíncrona - DC a 19.2K Baudios.
- Full-Duplex, Doble memoria intermedia en transmisión y recepción.
- Detección de errores - Paridad, Sobrecarga y Estructura.
- Compatible con los microprocesadores de Intel.
- Pakete DIL de 28 patillas
- Todas las entradas y salidas son compatibles TTL
- Se alimenta con +5V solamente.
- Reloj TTL.

El 8255A es una versión mejorada del estándar Intel 8251 USART, diseñado para comunicaciones de datos con las familias de microprocesadores de Intel como em MCS-48, 80, 85 e iAPX-86, 88. El 8251A se usa como dispositivo periférico y es programado por la CPU para que opere usando virtualmente cualquier técnica de transmisión serie usada actualmente (incluida la "bi-sync" de IBM). El USART acepta datos de la CPU en formato paralelo y los convierte en un flujo contínuo de datos serie para su transmisión. Simultáneamente, puede recibir datos en serie y convertirlos en caracteres paralelos para la CPU. El USART avisa a la CPU cuando está disponible para aceptar un nuevo carácter para transmitir o cuando ha recibido un carácter para la CPU. La CPU puede leer el estado completo del USART en cualquier momento. Esto incluye errores de transmisión de datos y señales de control como SYNDET, TxEMPTY. El chip se fabrica usando la tecnología de puertas de silicona N-canal.

Traducción de la hoja de características del chip 8251A

MC1488

QUAD LINE DRIVER

The MC1488 is a monolithic quad line driver designed to interface data terminal equipment with data communications equipment in conformance with the specifications of EIA Standard No. RS-232C.

Features:

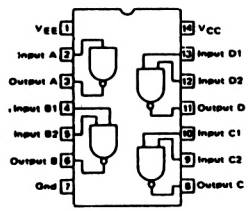
- Current Limited Output
  ±10 mA typ
- Power-Off Source Impedance
  300 Ohms min
- Simple Slew Rate Control with External Capacitor
- Flexible Operating Supply Range
- Compatible with All Motorola MDTL and MTTL Logic Families

QUAD MDTL LINE DRIVER
RS-232C
SILICON MONOLITHIC INTEGRATED CIRCUIT

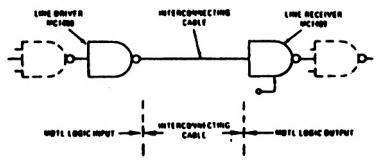
L SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116

P SUFFIX
PLASTIC PACKAGE
CASE 646

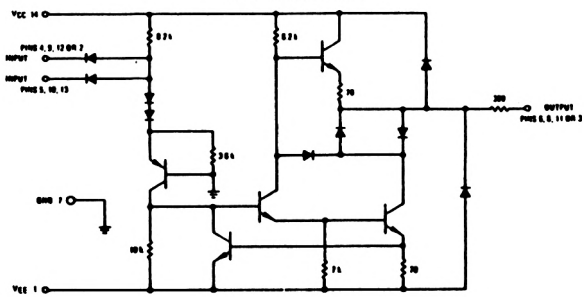
PIN CONNECTIONS



TYPICAL APPLICATION



CIRCUIT SCHEMATIC
(1/4 OF CIRCUIT SHOWN)



D89163R2

MC1488

CONTROLADOR CUADRUPLE DE LINEA

El MC1488 es un controlador monolítico cuádruple de línea, diseñadc para
interconectar equipos de terminales de datos con equipos de comunicaciones, conforme a las especificaciones EIA estándar No. RS-232C.

Características:

- Salida de corriente limitada, 10 mA.
- Impedancia fuente, apagado 300 Ohmios mínimo.
- Control simple de velocidad con condensador externo.
- Rango flexible de alimentación.
- Compatible con todas las familias lógicas de Motorola MDTL y MTTL.

Traducción de la hoja de características del chip MC1488

APENDICE CUATRO
Datos Varios:
Conexiones de las Patillas y Juego de Caracteres ASCII

Conexiones de las patillas de los conectores D para RS 232

Patilla	Nombre	Función		Sentido
1	FG	Frame Ground	- Masa del chasis	-
2	TD	Trasmited Data	- Datos trasmitidos	DTE a DCE
3	RD	Received Data	- Datos recibidos	DCE a DTE
4	RTS	Request To Send	- Petición para mandar	DTE a DCE
5	CTS	Clear To Send	- Borrar para mandar	DCE a DTE
6	DSR	Data Set Ready	- Grupo de datos preparados	DCE a DTE
7	SG	Signal Ground	- Masa de señales	-
8	DCD	Data Carrier Detect	- Detector pérdida de portad.	DCE a DTE
9	-	Positive DC test volt	- Tensión positiva de prueba	DCE a DTE
10	-	Negative DC test	- Tensión negativa de prueba	DCE a DTE
11	-	-	-	-
12	(S)DCD	Secondary DCD	- DCD secundario	DCE a DTE
13	(S)CTS	Secondary CTS	- CTS secundario	DCE a DTE
14	(S)TD	Secondary TD	- TD secundario	DTE a DCE
15	TC	Transmitter clock	- Reloj de trasmisión	DCE a DTE
16	(S)RD	Secondary RD	- RD secundario	DCE a DTE
17	RC	Receiver Clock	- Reloj receptor	DCE a DTE
18	RDC	Receiver dibit clock	- reloj 'dibit' receptor	DTE a DCE
19	(S)RTS	Secondary RTS	- RTS secundario	DTE a DCE
20	DTR	Data Terminal Ready	- Preparado terminal de datos	DTE a DCE
21	SQD	Signal quality detect	- Detector de calidad de señal	DCE a DTE
22	RI	Ring Indicator	- Indicador de autorización	DCE a DTE
23	DRS	Data Rate Select	- Selección velocidad datos	DTE a DCE
24	ETX	External Transmit clk	- Reloj externo de trasmisión	DTE a DCE
25	BSY	Busy	- Ocupado	DTE a DCE

DTE = Data Terminal Equipment - Equipo de terminal de datos

DCE = Data Comunications Equipement - Equipo de comunicaciones(por ejemplo MODEM)

NOTA

Debido a la adaptación de este "Estándar" durante un largo periodo de tiempo, los lectores pueden encontrar algunos terminales o equipos de comunicaciones que no usan algunas de las patillas como se ha indicado. Compruebe SIEMPRE el manual del equipo con el que se está conectando, antes de hacerlo.

Juego de caracteres ASCII

LSD:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
MSD:	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Códigos de control:

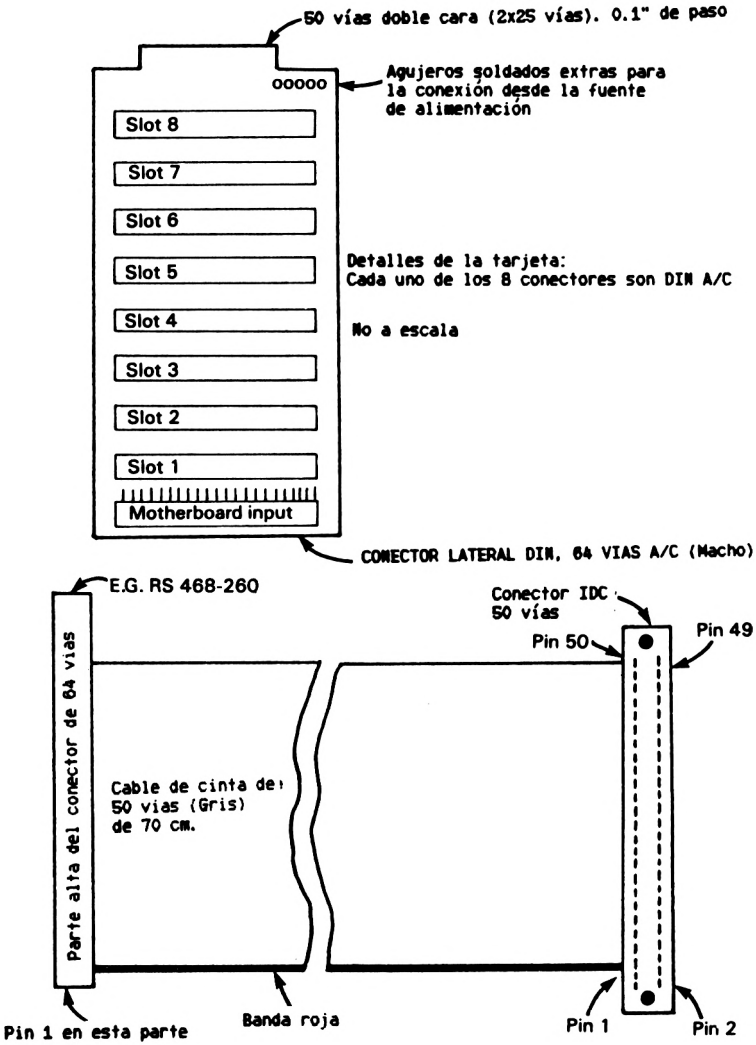
NUL=Carácter de relleno
SOH=Principio de cabecera
STX=Principio de trasmisión
ETX=Fin de texto
EOT=Fin de trasmisión
ENQ=Petición
ACK=Reconocimiento
BELL=Señal audible
BS=Espacio hacia atrás
HT=Tabulador horizontal
LF=Salto de línea
VT=Tabulador vertical
FF=Avance de página
CR=Retorno de carro
SO=Desplazar
SI=Reunir
DLE=Escape de conexión
DC1-DC4=Control de dispositivo 1-4
NAK=Reconocimiento negativo
SYN Carácter de sincronismo
ETB=Fin bloque trasmisión
CAN=Cancelar
EM=Fin de medio
SUB=Sustituir
ESC=Escape
US=Separador de unidad
SP=Espacio
DEL=Borrar

APENDICE CINCO
'Bus' de Expansión del CPC

EL 'BUS' DE EXPANSIÓN EXTENDIDO:

La expansión del 'bus' de 50 contactos se encuentra en un conector marcado como "Floppy disc" en la parte trasera de su CPC. El 'bus' de expansión tiene todas las señales requeridas para añadir todo tipo de periféricos a un ordenador básico. Esto incluye la línea de alimentación de 5 voltios, conexiones de masa, y la señal del reloj maestro que se usa para el microprocesador Z80A. Sin embargo se pueden presentar algunos problemas cuando necesitamos conectar un número apreciable de periféricos al 'bus' de expansión. Cuando tenemos periféricos que necesitan tensiones de alimentación como +12V, o -12V, el 'bus' de expansión del CPC no puede suministrarlas - su fuente de alimentación (PSU) solo produce +5 voltios. Esto significa que una tarjeta de expansión, como un interface serie RS232, debe tomar sus +12V y -12V de otra fuente distinta al 'bus' de expansión. Otro problema se presenta porque, aunque hay dos líneas de masa entre las 50 del 'bus' de expansión, solo la línea 27 se usa para suministrar los +5V desde el ordenador a los dispositivos externos. La conexión entre el CPC y el 'hardware' externo se puede hacer mediante un cable de cinta estándar que soporte 1 amperio por conductor. Cuando se conectan varias tarjetas externas, este amperaje se puede sobrepasar fácilmente, haciendo que se sobrecargue la línea 27. El resultado será que obtendrá en los chips de las tarjetas externas menos de los 5 voltios. Esto a su vez dará como resultado una operación inestable, en el mejor de los casos, o una falta total de operatividad, en el peor.

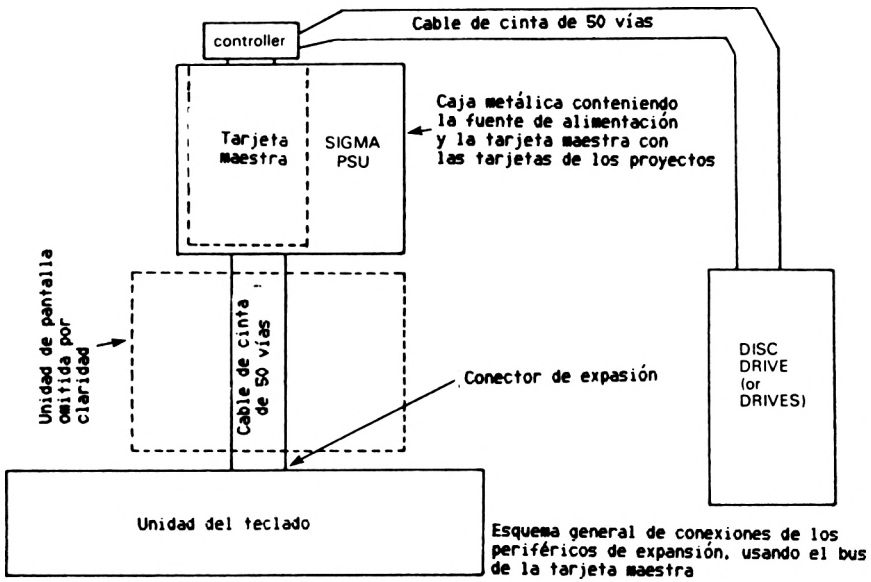
El desarrollo de los proyectos detallados en este libro nos ha llevado a diseñar una tarjeta maestra de 64 líneas. Esta tarjeta maestra la distibuye 'PCB suppliers' - ver página tres. La tarjeta maestra acomoda hasta ocho conectores para otras tantas tarjetas de expansión. El 'bus' de 50 contactos atraviesa la tarjeta maestra y aparece disponible en el extremos exterior, permitiéndole que conecte también una unidad de disco o un MODEM. Se ha seleccionado un 'bus' de 64 conexiones por diversas razones, la principal es permitir la conexión en el mismo 'bus' de las señales del conector de 50 contactos, junto con las de las nuevas tensiones, usando las 14 líneas restantes. Un beneficio adicional es que permite también usar varios conductores para las tensiones y las masas. Estas tensiones extra de alimentación, y una fuente extra de +5V, las proporciona la fuente de alimentación independiente. Los detalles sobre esta fuente se dan al principio del capítulo tres. Las figuras AP5.1, AP5.2 y AP5.3 muestran el diagrama de conexiones de todas las unidades y componentes implicados. Estos son el CPC, la fuente de alimentación externa, la tarjeta maestra y la unidad de disco Amstrad DD1 (si la tiene).



Detalles del cable para conectar la puerta de expansión a la tarjeta

Fig. AP5.1

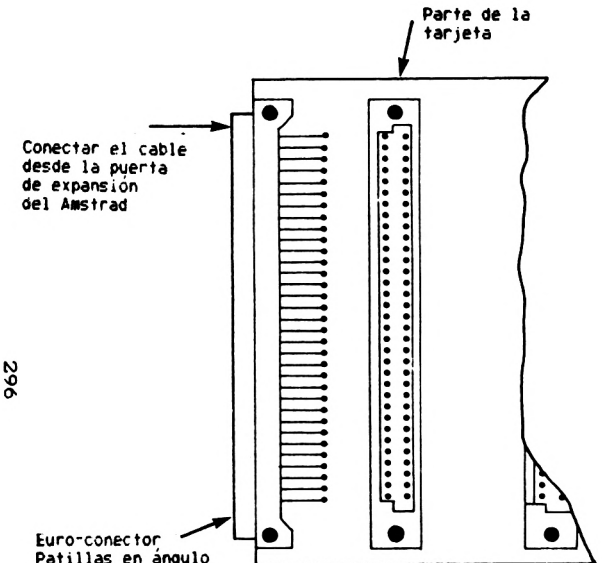
Fig AP5.2



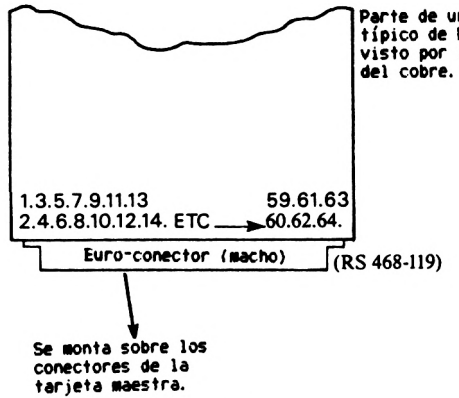
Esquema general de conexiones de los periféricos de expansión, usando el bus de la tarjeta maestra

La figura AP5.4 muestra la designación de las conexiones del 'bus' expandido. Como puede ver todavía quedan dos líneas libres. Estas se han situado de forma que sirvan como medida de protección contra cortocircuitos accidentales entre las líneas de +12 y -12 voltios. Si se conectan accidentalmente a algo que funciona normalmente con +5 voltios, puede producir un daño irremediable - así que tenga mucho cuidado. Las conexiones 61 y 62 llevan una tensión llamada +VDC. Esta tensión está sin regular pero rectificada y se usa para alimentar los reguladores de tensión situados en las tarjetas de los proyectos, si se requiere. La conexión de +VDC puede dañar a los dispositivos de +5V, así que tenga mucho cuidado.

Figura AP5.3



Vista superior de la tarjeta maestra, mostrando la orientación de un Euro-conector. (Los demás se sitúan de la misma forma)



Vista de un proyecto tipo por la cara de las soldaduras, mostrando las patillas del Euro-conector, se numeran usando este esquema. NOTA: Difiere de los números marcados en el conector

Fig AP5.4

Pat.	Designación	Pat.	Designación	Pat.	Designación
1	SOUND	22	D4	43	ROMDIS
2	GROUND	23	D3	44	RAMRD
3	A15	24	D2	45	RAMDIS
4	A14	25	D1	46	CURSOR
5	A13	26	DO	47	LPEN
6	A12	27	+5V	48	EXP
7	A11	28	MREQ	49	GROUND
8	A10	29	M1	50	Reloj 4 MHz
9	A9	30	RFSH	51	+5V (Ext)
10	A8	31	IORQ	52	+5V (Ext)
11	A7	32	RD	53	50 Hz TTL (EXT)
12	A6	33	WR	54	libre
13	A5	34	HALT	55	+12V (Ext)
14	A4	35	INT	56	+12V (Ext)
15	A3	36	NMI	57	-12V (Ext)
16	A2	37	BUSRQ	58	-12V (Ext)
17	A1	38	BUSAK	59	libre
18	AO	39	READY	60	DOUT
19	D7	40	BUS RESET	61	+VDC (Ext)
20	D6	41	RESET	62	+VDC (Ext)
21	D5	42	ROMEN	63	GROUND
				64	GROUND

Muchas de las señales del 'bus' se toman directamente desde el microprocesador Z80A. Sin embargo hay unas señales específicas del CPC cuya función se explica a continuación.

BUS RESET - 40: Es una entrada para conectarla a un interruptor externo. El interruptor se debe conectar entre BUS RESET y GROUND (2, 49,63 o 64). Cuando se pulsa este interruptor, se restaura el ordenador completo, y cuando se suelta el CPC rearranca como si se hubiera encendido de nuevo.

RESET - 41: Es una salida del circuito de Restauración de encendido (POR - Power On Reset) dentro del CPC. Se debe conectar a las entradas de RESET de los chips de las tarjetas conectadas al 'bus' de expansión, para asegurarse de su correcta inicialización a tiempo de encendido.

ROMEN - 42: Es una salida de los circuitos de decodificación de direcciones de la matriz de puertas de vídeo. Se pone en estado lógico bajo cuando el 'bus' de direcciones contiene una dirección aplicable a la ROM superior. Su rango es desde Hex COOO a FFFF.

ROMDIS - 43: Es una entrada que, cuando se pone en estado lógico alto, inactiva el chip de la ROM del CPC. Como el chip de la ROM está desactivado totalmente, no se puede acceder ni a la parte baja ni a la alta de la ROM.

RAMRD - 44: Cuando se pone esta salida en estado bajo, la matriz de puertas de vídeo ha abierto la memoria intermedia entre la RAM del CPC y el 'bus' de datos. En otras palabras un estado bajo en esta salida significa que el Z80A está leyendo de la memoria RAM. La si-

guiente señal - RAMDIS, elimina a RAMRD.

RAMDIS - 45: Es una entrada que, cuando se pone en estado bajo, evita que el contenido interno de la RAM esté disponible en el 'bus' de datos. Esto permite al usuario implementar esquemas donde las ROM, conteniendo programas de usuario, se puedan conmutar dentro del mapa de memoria, para reemplazar la RAM. Es una facilidad muy útil para una aplicación que requiera varias ROM, pero solamente un pequeño espacio en RAM.

CURSOR - 46: Es una conexión a la patilla de activar el cursor del chip controlador de VDU HD6845. Su función se explica en la hoja de datos reproducida en el capítulo tres.

LPEN - 47: Es una conexión de entrada para el lápiz óptico. Es una conexión directa al chip controlador de VDU HD6845. Vea el apéndice tres si quiere una descripción completa de los detalles.

EXP - 48: La función de esta entrada no está definida claramente en ninguna documentación del 'hardware' de Amstrad. Parece que es un bit sensor, para decirle al sistema operativo que necesita saber si hay algo conectado en el 'bus' de expansión. Está conectado al bit 5 de la puerta B del PPI P8255A interno del CPC. (La dirección de E/S de la puerta B es Hex F5xx).

50Hz TTL - 53: Esta señal se crea en la fuente de alimentación exterior. Se toman los 50 ciclos exactos de la red. Se suministra para el uso en los proyectos de los lectores.

DOUT - 60: Esta es una de las señales que esperamos que nunca tendrá que usar. Se suministra para que se puedan conectar regeneradores de señal al 'bus' de expansión. DOUT invierte el flujo de los datos en el 'bus' de datos. Solo se necesita si tiene que conectar un circuito regenerador de señal en el 'bus'.

Todas las señales de la lista que tiene la palabra Ext entre paréntesis, se originan en, o son líneas que vienen de, la fuente de alimentación externa, - vea el principio del capítulo tres.

El 'bus' de expansión consiste físicamente en la fuente de alimentación externa y la tarjeta maestra de expansión del 'bus'. Deben construirse dentro de una caja de metal. Como se ve en el diagrama del capítulo tres, la caja se debe conectar a masa por razones de seguridad. La conexión de la caja a masa puede aumentar la inmunidad a los ruidos y eliminar las interferencias en los receptores de radio o televisión. Vea la página iii para detalles de los suministradores de la tarjeta maestra y de los circuitos impresos de las otras tarjetas.

Direcciones del 'bus' de expansión de los proyectos detallados en este libro

Hex Dirección E/S	Usada para	Nombre proyecto
DF00	Registro selección de ROM	Tarjeta exp.
F8E0	Dirección local (baja)	Programador EPROM
F8E1	Dirección local (media)	Programador EPROM
F8E2	Registro datos lectura/escritura	Programador EPROM
F8E3	Registro de control PPI	Programador EPROM
F8E4	Registro de estado del programador de EPROM	Programador EPROM
F8F0	PPI #1 puerta A	Tarjeta Multiproy
F8F1	PPI #1 puerta B	Tarjeta Multiproy
F8F2	PPI #1 puerta C	Tarjeta Multiproy
F8F3	PPI #1 registro de control	Tarjeta Multiproy
F8F4	PPI #2 puerta A	Tarjeta Multiproy
F8F5	PPI #2 puerta B	Tarjeta Multiproy
F8F6	PPI #2 puerta C	Tarjeta Multiproy
F8F7	PPI #2 registro de control	Tarjeta Multiproy
F8F8	Registro alófonos sintetizador de palabra	Sintetizador voz
F8F9	Registro de inflexión	Sintetizador voz
F9E0	USART 1 registro de datos	RS232 Versión B
F9E1	USART 1 registro de control	RS232 Versión B
F9E2	USART 2 registro de datos	RS232 Versión B
F9E3	USART 2 registro de control	RS232 Versión B
F9E4	USART 3 registro de datos	RS232 Versión B
F9E5	USART 3 registro de control	RS232 Versión B
F9E6	USART 4 registro de datos	RS232 Versión B
F9E7	USART 4 registro de control	RS232 Versión B
F9E8-F9EB	Registro velocidad trasmisión canales 0 y 1	RS232 Versión B
F9EC-F9EF	Registro velocidad trasmisión canales 2 y 3	RS232 Versión B
FAE0	Dirección local (baja)	Disco en ROM
FAE1	Dirección local (media)	Disco en ROM
FAE2	Dirección local (alta)	Disco en ROM
FAE3	Control chip PPI del disco en ROM	Disco en ROM
FAE8	Registro lectura de datos del disco en ROM	Disco en ROM

APENDICE SEIS
Empiece Aquí

PARA LOS LECTORES NUEVOS EN ORDENADORES, O NUEVOS EN 'HARDWARE' DE ORDENADORES.

Binario: Decodificador de direcciones: Chips lógicos: Esquemas de circuitos: Proceso físico de construcción: Soldadura: Trampas: Consideraciones generales: Fuentes de alimentación - "¡Basta ya!,¡Es demasiado!".

Si usted no ha construido antes un circuito, su primer intento puede ser una experiencia frustrante. Mi primer intento terminó llamando a un amigo que probó el circuito durante unos minutos y eliminó un pequeño trozo de soldadura que había caído entre dos patillas de un chip. Después funcionó perfectamente. Esta historia nos enseña dos cosas. La primera es que cuando diseñe y construya un proyecto, lo mas normal es que no funcione a la primera. (No se apure cuando construya diseños de las revistas o libros, hay una posibilidad muy remota de que funcione a la primera). La segunda, siempre suele haber una razón simple para que el proyecto se niegue a funcionar. Es muy raro que sea por un chip estropeado.

Intentemos eliminar algo de confusión en la construcción de circuitos, respondiendo a algunas de las preguntas que se le puedan ocurrir cuando mire los esquemas y las explicaciones de los proyectos de este libro. Antes que nada, algunos conceptos básicos, el arte de direccionamiento de memoria y de los periféricos.

El microprocesador Z80A del CPC 464/664 es capaz de acceder a 65535 posiciones de memoria. Imagínese esto como 65536 casillas, como las que puede encontrar en una oficina de correos. Cada casilla tiene asignado un número único de identificación, a este número se le denomina dirección. Este primer juego de casillas se utiliza como direcciones de memoria, y se usan para almacenar valores, caracteres o listas de instrucciones que forman un programa. El Z80A puede acceder además a un segundo grupo de 65536 casillas. El segundo grupo se usa para Entrada/Salida. (Se suele abreviar como E/S). Podemos decir, sin embargo, que el segundo grupo de casillas no tienen fondo, y que los caracteres o valores que colocamos en ellas salen fuera de la oficina de correos hacia el mundo exterior. De forma similar, los caracteres o valores que vienen desde el mundo exterior pueden entrar dentro del ordenador por medio de este segundo grupo de casillas. En teoría, el Z80A puede leer datos desde, o escribir dentro, de cualquiera de las casillas de cualquiera de los grupos. En la práctica depende del chip físico al que está conectado cada una de las casillas. Algunas de estas casillas son ROM (Memoria de solo lectura), mientras que otras son memoria de lectura y escritura. (Conocidas como Memoria RAM).

Cuando el Z80A necesita acceder a una posición particular (o casi-

silla) en su memoria, manda una representación binaria de la dirección de esa posición dentro de una cosa que se llama 'bus'. (El concepto de binario se explicará enseguida). El 'bus' de direcciones es un grupo de 16 patillas del chip Z80A. Se numeran de A0 a A15. Las 16 señales que salen desde estas patillas se conocen colectivamente como líneas del 'bus' de direcciones. Estas líneas del 'bus' de direcciones se extienden a todas las partes del ordenador. Por medio de pistas del circuito impreso del CPC (PCB), o por medio del conector de expansión, a dispositivos externos, como los detallados en este libro, o al controlador de discos de Amstrad, por ejemplo. Cada banco de memoria, o dispositivo externo tiene un decodificador de direcciones asociado. La misión del decodificador de direcciones es analizar constantemente cada dirección mandada desde el Z80A. Cada decodificador detecta una dirección concreta, o un rango de direcciones. Cuando el decodificador detecta una dirección, produce una señal para activar el chip de memoria, o el dispositivo periférico con el que está asociado. A esta señal que produce el decodificador de direcciones se le suele llamar señal de 'Chip Select' (Selección de chip - CS). Después de que se ha producido la señal de selección de chip, tiene lugar la transferencia de datos desde el Z80A a la memoria o el periférico, o viceversa, sobre una cosa que se llama 'bus' de datos. El 'bus' de datos es un grupo de líneas de señal sobre las que se realiza la trasferencia de datos desde y hacia la memoria, Z80A y periféricos. El 'bus' de datos, como el de direcciones, está conectado a todas las partes del ordenador.

Ya hemos mencionado anteriormente el término binario, veamos ahora el sistema de numeración binario, y por qué es tan importante en informática. (Vea también el apéndice dos del manual de instrucciones de su Amstrad).

Todos estamos condicionados a contar en potencias de 10. Nuestra moneda, sistema de pesos y medidas, suma, resta, multiplicación y división aritmética, se cuanta en potencias de diez. Si yo le digo que he gastado 700 horas hombre en preparar este libro, usted sabrá inmediatamente que quiero decir: ningún millar, siete centenas, ninguna decena y ninguna unidad. Pero ¿si le digo que he tardado 1010111100 horas?. No me creería, porque usted asumiría razonablemente que el número anterior era un valor decimal, y que significaría que había tardado 32 años en preparar este libro. Obviamente, habría necesitado un poco más de agudeza mental. Pero, si mira de nuevo la cifra anterior, verá que solamente contiene dos de los diez números que se suelen usar en nuestro esquema numérico habitual, que son el uno y el cero. Como ya habrá imaginado, el número no estaba expresado en decimal, sino en binario.

Comparemos los dos números, 700 y 1010111100. Ya hemos dicho que 700 decimal significa siete centenas, ninguna decena y ninguna unidad. Veamos como se desglosa el número 1010111100.

En binario solo se usan dos números. Por esto es por lo que se usa el sistema binario en los ordenadores. Se acopla de forma ideal, porque las señales de los ordenadores pueden ser activas - tensión presente (uno lógico), o inactivas - tensión ausente (cero lógico). Los unos y los ceros se usan para expresar valores numéricos, de la misma forma que los diez números usados en el sistema decimal. Como en el sistema decimal, la posición que ocupa cada dígito le da su peso. (El peso es la contribución sobre el total). Si empezamos por la derecha del número decimal, esto es, por la columna de las unidades, y nos

movemos hacia la izquierda a través de las otras columnas, veremos que cada columna sucesiva tiene un peso que es diez veces superior a la anterior. Así, nos movemos de las decenas a las centenas (10 x 10 = 100), y de las centenas a los millares, y así sucesivamente. En el sistema de numeración binario la columna de más a la derecha tiene un peso de uno, y a medida que nos movemos hacia la izquierda, cada columna sucesiva tiene un peso que es dos veces el de la anterior. Así, los valores de las columnas desde la derecha a la izquierda son: 1,2, 4,8,16,32,64,128 y así sucesivamente. Para ilustrar esto, escribamos nuestro 700, y junto a él pongamos el ejemplo binario - después sumemos todos los pesos individuales presentes es cada número:

Centenas	*Decenas*	*Unidades*	*512*	*256*	*128*	*64*	*32*	*16*	*8*	*4*	*2*	*1*
7	0	0	1	0	1	0	1	1	1	1	0	0

```
7 x 100 = 700
0 x  10 = 000
0 x   1 = 000
          -----
  TOTAL     700
          -----
```

```
0 x   1 = 000
0 x   2 = 000
1 x   4 = 004
1 x   8 = 008
1 x  16 = 016
1 x  32 = 032
0 x  64 = 064
1 x 128 = 128
0 x 256 = 256
1 x 512 = 512
          -----
  TOTAL     700
          -----
```

Con este ejemplo puede ver fácilmente como funciona el sistema de numeración binario. Pero no se lo crea porque yo se lo digo, pregúntele a su Amstrad. Consiga el mensaje "Ready", después teclee "PRINT BIN$(700)". Esto le dice al Locomotive BASIC que imprima el valor en binario del número decimal que aparece entre paréntesis. El BASIC también puede hacer lo opuesto, esto es obtener el valor decimal de un número binario, tecleando "PRINT STR$(&X1010111100)". Inténtelo y lo verá por usted mismo.

Volviendo a las señales del 'bus' de direcciones del Z80A, estas líneas pueden tener o no tensión, para informar a los dispositivos conectados al 'bus' de direcciones a qué chip quiere acceder. Estas señales forman el número binario que representa la dirección. No intente ver las señales del 'bus' de direcciones con un aparato de medida, aparecen y desaparecen muy rápidamente, cada una está presente solamente una fracción de milésima de segundo antes de que sea reemplazada en el 'bus' por la siguiente dirección a la que necesita acceder el Z80.

¡¡Más binario!! Si, pero esta vez veremos como simplificar la escritura de números binarios. Esa gran cantidad de ceros y unos encadenados son horriblemente tediosos de escribir, y se pueden cometer muchos errores. Afortunadamente, raramente necesitará convertir un número en su representación por unos y ceros. Esto se debe al uso extendido del sistema de notación hexadecimal. Se le llama hexadecimal porque usa el juego de dígitos decimales, del cero al nueve, y también las seis primeras letras del alfabeto. La notación hexadecimal

nos proporciona una forma de escribir cualquier combinación de ceros y unos, que se puedan encontrar en cuatro columnas binarias, como un carácter simple. La lista completa es:

BINARIO	*HEXADECIMAL*	*DECIMAL*
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Cuando un número binario tiene más de cuatro columnas, se dividen los unos y los ceros en dos grupos de cuatro, desde la derecha, por ejemplo, el valor binario 1010 0001 se escribirá en HEXADECIMAL,(de ahora en adelante usaremos HEX para acortarlo) como, HEX A1, mientras que el binario 0100 1110 se escribirá como HEX 4E. Observe como hemos separado los dos grupos de cuatro números binarios con un espacio, para mejorar la legibilidad. Observe también que cuando se escribe un número en Hex, debe dejar siempre claro que es un número Hex, y no uno decimal. Esto se hace escribiendo o "Hex CE" o "XCE". En el contexto del BASIC del CPC, el signo "&" se usa también para indicar que el número que sigue es un número hexadecimal. Los códigos de cada carácter Hex parecen difíciles de aprender, pero después de algunas semanas de uso resultan familiares - puede pasarle, como me ha pasado a mi, que después de mucho trabajo en Hex, alguna vez intenta hacer las cuentas de la tienda en Hex - no se lo recomiendo, puede confundir a las cajeras.

Ahora ya tenemos una forma de expresar como direcciones, estos unos y ceros que salen del Z80A. Recuerde que el 'bus' de direcciones contiene la dirección de la posición de memoria o de la puerta de E/S (o si usa la analogía que acabamos de ver, el casillero), a la que quiere acceder el Z80A. La transacción real de datos entre la posición direccionada y el Z80A tendrá lugar sobre el 'bus' de datos.

Más adelante, en este capítulo, veremos un chip decodificador de direcciones, pero antes debemos cubrir algunos conceptos básicos de los chips en general.

Se da el nombre de "chip" a cualquier circuito integrado. Espero que sepa que un chip puede contener hasta medio millón de transistores, embutidos todos dentro de un pequeña pieza de silicona. El número exacto de transistores contenidos en un chip determinado depende de lo compleja que sea su función. El chip real está encerrado en una envoltura cerámica, y las pequeñas conexiones alrededor del chip se extienden hacia las patillas de la envoltura. El número de patillas de una envoltura de un chip (que llamaremos en adelante simplemente

chip) dependerá de cuántas conexiones externas necesita para funcionar. Los números comunes de patillas son los siguientes:

8 patillas	14 patillas	16 patillas
18 patillas	20 patillas	24 patillas
28 patillas	40 patillas	64 patillas

Ya que las patillas está dispuestas en dos ileras a cada lado del chip, se les ha dado el nombre de paquete Dual In Line (Doble en línea). Este nombre se abrevia como DIL. Tanto los vendedores como los fabricantes hacen referencia a los chips como suministrados en paquete DIL de X patillas. Cuando vaya a conectar realmente un chip en un circuito, tendrá dos formas de hacerlo. La primera, soldándolo directamente en su lugar, soldando cada patilla del chip a la tarjeta para hacer la conexión. Este sistema es bueno si puede soldarlas bien y rápidamente. Pero si no puede hacerlo, puede dañar el chip con la aplicación de excesivo calor por un periodo de tiempo demasiado largo. Otra desventaja de este sistema es que el chip puede fallar en algún momento, y desmontarlo significará aún más uso del soldador, que puede llegar a dañar a otro dispositivo cercano o a la misma tarjeta.

También puede dañar fácilmente las pistas del circuito impreso. Un método ligeramente más caro, que elimina los problemas mencionados, es usar zócalos para los chips. Estos zócalos están disponibles en tamaños de paquete DIL. Los zócalos absorben el calor del soldador. Si alguna vez falla el chip, se puede sustituir fácilmente sacándolo del zócalo y metiendo un chip nuevo. El costo extra de los zócalos suele estar ampliamente justificado.

Ahora que ya conoce lo que es un chip, ¿o todavía no? Me temo que hará reír a los vendedores de componentes electrónicos si les dice simplemente que quiere un chip. Puede ser similar a meterse en una floristería y pedir una flor.

Los tipos de chips se especifican por combinaciones de números y letras. Esta identificación se obtiene de la lista de componentes del proyecto. Cuando vaya a diseñar sus propios circuitos, deberá conseguir un gran libro, publicado por los fabricantes de chips, que detalla todos los productos de ese fabricante. Algunos de estos libros le darán detalles también de aplicaciones de los chips. Aquí haremos una pequeña simplificación sobre los tipos de chips. Los chips que son menos complejos que los microprocesadores, o chips de voz, etc, son chips SSI o MSI. (¡Más iniciales!). Estas significan Small Scale Integration - Integración a pequeña escala - y Medium Scale Integration - integración a media escala). Las funciones de los chips de este tipo suelen ser contadores, puertas lógicas, decodificadores, y registros de desplazamiento. Veremos este tipo de chips un poco más adelante. Los chips SSI y MSI está disponibles a bajo precio, desde unas 75 pesetas hacia arriba. Por supuesto, el precio crece con la complejidad del propio chip - aunque también depende del uso del mismo, un chip que se use mucho en circuitos comerciales necesita tiradas muy grandes, lo que abarata los costos y por tanto el precio final.

Existen en dos familias lógicas. Esto significa que puede obtener chips que hagan las mismas funciones pero que estén fabricados con procedimientos totalmente distintos. Veamos estas dos familias.

La primera familia se llama TTL (Aún más iniciales. Estas signifi-

can Transistor Transistor Logic - Lógica transistor transistor). La TTL fue la primera familia de chips lógicos, y estuvo disponible a partir de 1970. Dentro de la familia TTL hay muchas variantes. La familia estándar de chips TTL tiene números que comienzan por 74 - como 74138 que es un decodificador de direcciones TTL. Texas Instruments, que comercializó los primeros chips TTL, ha lanzado nuevas subfamilias TTL que tienen muchas mejoras, como mayor velocidad y menor consumo de potencia. La principal de estas subfamilias es el rango 74LS. LS significa Low power Schottky. (Schottky de baja potencia). El rango de chips LS se pueden conectar, y funcionan, de la misma forma que los originales del rango 74, pero consumen generalmente menos potencia, y pueden ser más rápidos. Así, el 74LS138 puede reemplazar al 74138, o un 74LS32 puede reemplazar a un 7432. Los chips TTL solamente pueden operar con una alimentación de +5V, y son totalmente interconectables unos con otros. La versión LS de TTL no son, probablemente, más usados que la segunda familia de chips que veremos ahora.

La segunda, y ligeramente más nueva, familia de chips lógicos se llama familia CMOS. (CMOS significa Complementary symmetry Metal Oxide Semiconductor - Semiconductor de oxido metálico de simetría complementaria). Hay un par de subfamilias de CMOS. Los chips originales CMOS se llaman series "A". Fueron poco populares inicialmente por su vulnerabilidad a daños por electricidad estática. Esto significa que cualquiera que llevara puesto un jersey de lana y manejara chips de la serie "A", tendría muchas probabilidades de dejar el chip inoperativo por el simple hecho de cogerlo con la mano. Los fabricantes recomendaban a las personas que tuvieran que manipulas chips CMOS, que antes de coger los chips con una mano, tuvieran cogido con la otra un cable conectado a tierra, para eliminar la electricidad estática del cuerpo. Afortunadamente RCA, que fabricó los primeros CMOS, lanzó la serie "B". Esta segunda serie probó que era más robusta, y mi experiencia con chips de la serie "B" me demostró que se pueden tratar de la misma forma que los de la familia TTL, sin resultados adversos. Por ahora debe preguntarse cuales son las ventajas de esta segunda familia. Son varias. La primera es su bajísimo consumo. Muchos chips CMOS tienen consumos de corriente que se pueden medir en micro amperios, esto los hace ideales para las operaciones alimentadas por baterías. La segunda, y más importante facilidad de los chips CMOS es que pueden operar con varas tensiones de alimentación distintas, normalmente entre 5 voltios y 18 voltios. Esto permite más flexibilidad en el diseño de fuentes de alimentación, y permite la operación directa desde fuentes de alimenatación de automoción etc. Por último, hay algunas funciones lógicas en CMOS que no se encuentran en un solo chip de la familia TTL.

Respecto a la numeración, Toda la familia de chips lógicos CMOS empiezan por un 4 y se puede distinguir la serie "A" de la serie "B" por el sufijo que aparece después del número del chip. Así, un 4040A es una versión de chip divisor de 12 pasos de la serie "A", y un 4040B es lo mismo pero de la serie B. Como las versiones estándar y LS de los chips TTL, las series "B" de chips CMOS pueden reemplazar los chips de la serie "A" en el 99.99 por ciento de todas las aplicaciones de los circuitos lógicos. En los proyectos de este libro no he especificado que use los dispositivos de la serie "B", porque ya he avisado que no use nunca los de la serie "A" a menos que el diseño lo diga específicamente. Ninguno de los diseños de este libro necesitan la serie "A". CMOS no ha llegado a ser tan popular como TTL en el

mercado de los aficionados. Esto viene , creo, del prejuicio creado por el problema de la serie "A". Además los CMOS suelen costar un poco más caros que los equivalentes de la familia TTL.

Las dos familias no son incompatibles. Suponiendo que usted use una fuente de alimentación de +5v con CMOS, puede interconectar CMOS y TTL sin problemas de compatibilidad, sujeto a restricciones de carga, - pero de eso hablaremos en otro libro.

Veamos ahora las funciones lógicas reales. No profundizaremos mucho en este tema, solo lo suficiente para darle unos pequeños conocimientos.

Recuerde que en todas las descripciones de las funciones lógicas, cuando decimos lógica alta o uno lógico, queremos decir que la tensión en el punto indicado es de uno 4.5 a 5 voltios. Cuando decimos lógica baja o cero lógico, queremos decir que la tensión en ese punto es menor de 1 voltio.

Las funciones lógicas más simples se llaman puertas. Como son muy simples, encontrará hasta cuatro o cinco funcionalmente idénticas dentro de un mismo chip, aunque con entradas y salidas independientes. Esto es una puerta. Una puerta es la forma de incluir la capacidad de una decisión simple dentro de un circuito. Las puertas típicas tienen varias patillas de entrada y una sola de salida. Hay cuatro tipos básicos de puertas. Tiene los siguientes nombres:

NAND AND OR NOR

Estos nombres que suenan tan esotéricos abarcan varios conceptos simples. Veamos por ejemplo la puerta AND. Como ya dijimos anteriormente, una puerta AND, como otra cualquiera, tiene varias entradas, pero una sola salida. En este ejemplo vamos a analizar la puerta AND de dos entradas. La puerta AND tiene la capacidad de decirnos cuándo las dos entradas tienen aplicados unos lógicos. Esto lo hace dándonos un uno lógico a la salida cuando ambas entradas está con uno lógico. Si alguna de las dos entradas tiene un cero lógico, la salida de la puerta AND permanecerá con lógica baja. Esto es válido hasta en una puerta AND con ocho entradas. La salida solo se pondrá con uno lógico cuando las ocho tengan uno lógico aplicado a ellas. De lo anterior se deduce fácilmente lo que hace la puerta OR. La salida de la puerta OR se pondrá con uno lógico cuando una o más de las entradas están con uno lógico. Así, para una puerta OR de cuatro entradas: si la entrada uno o la entrada dos, o la entrada tres, o la entrada cuatro tiene uno lógico, la salida tendrá el mismo nivel.

El funcionamiento de las puertas NAND y NOR es también muy simple. La N que aparece delante nos dará la pista. Las condiciones de entrada necesarias para activar la salida son las mismas que para las puertas AND y OR. La única diferencia es que en lugar de ponerse baja la salida cuando no cumple las condiciones de entrada, y ponerse alta cuando las cumple (todas las entradas altas en la puerta AND y cualquiera de las entradas alta en la puerta OR), la salida de las puerta NAND y NOR están normalmente a uno lógico, pero se ponen a cero cuando se cumplen las condiciones de entrada. Así, la salida de una puerta NAND permanecerá con uno lógico hasta que todas las entradas se pongan con uno lógico, en cuyo caso la salida se pondrá a cero lógico. La salida de la puerta NOR se pondrá a cero cuando una o más de

las entradas estén a uno.

Entonces, ¿para qué se usan estas puertas?. Individualmente no pueden hacer mucho, pero cuando se interconectan varias de ellas, el resultado puede ser muy útil. Como demostración vamos a ver su CPC. Tiene muchas puertas en su interior, aunque muchas de ellas están situadas en el chip microprocesador Z80A. En términos generales, las puertas lógicas están interconectadas dentro del Z80A para cerrar y abrir los caminos de los datos, basándose en las condiciones que aparecen en sus entradas. El Z80A no podría existir sin puertas lógicas. A un nivel más modesto, podrá ver que los proyectos de este libro hacen uso libre de puertas individuales conectadas entre si. Entonces, ¿qué apariencia tiene una puerta lógica?, ¿cómo se puede saber las que contiene un chip?. Resulta extraño que haga estas preguntas, pero veamos ahora un chip.

La figura AP6.1 muestra la visión en planta de un chip TTL llamado 74LS08. (Recuerde que el 7408 tiene la misma distribución, pero consume más potencia). Puede ver que hay 14 patillas en el chip, 7 a cada lado. Así que podemos decir inmediatamente que es un paquete DIL de 14 patillas. Las patillas están numeradas. Hay una regla general que dice que la patilla más cercana a un pequeño agujero redondo, o una pequeña marca, en la cubierta plástica del chip, es SIEMPRE la patilla número 1. Las demás patillas se numeran hacia abajo en un lado y hacia arriba en el otro, como se ve en este 74LS08. Esta convención se aplica a TODOS los chips, no he visto ninguno que no siga esta regla. Lo siguiente que tenemos que notar son las dos patillas de las esquinas, la 14 y la 7. La patilla 14 se llama VCC. En inglés significa Positive supply (alimentación positiva), que para los chips TTL es de +5V. Así que esta patilla toma la tensión de alimentación para que funcionen todos los elementos de este chip. La patilla 7 se llama GND. En inglés significa GrouND (masa o tierra, OV) que es la conexión de masa o de cero voltios de la fuente de alimentación. Ahora veamos lo que estábamos explicando. Las patillas 1 y 2 entran dentro de un símbolo extraño, que es realmente el símbolo que representa a una puerta lógica AND. La salida de esta puerta AND en particular está conectada a la patilla 3 del chip. Como puede ver, en este chip hay en total cuatro puertas AND de dos entradas cada una. Cada una de ellas opera totalmente independiente de las otras, lo único que comparten son la envoltura en la que viven y la tensión de alimentación que entra por las patillas 14 y 7. En todos los aspectos cada puerta AND es totalmente independiente de las otras. Este no es el lugar apropiado para poner experimentos que pueda usted hacer, así que créame si le digo que si conecta las patillas 1 y 2 a la patilla 14, la patilla 3 se pondrá a nivel lógico uno. Mientras que si conecta la patilla 1 o 2 a la patilla 7. Obtendrá un nivel lógico cero en la patilla 3. El 74LS08 se vende como un paquete de cuatro puertas AND de dos entradas. Tanto en TTL como en CMOS hay chips que contienen puertas AND con dos, tres o cuatro entradas.

Acabamos de ver el símbolo de una puerta AND, podrá encontrar una lista completa de los símbolos de los otros tipos de puertas lógicas en el apéndice 1 de este libro. Tenga en cuenta que hay otros símbolos, usados por otras personas, para representar las mismas cosas, aunque los que se usan en este libro son probablemente los más usados.

Fig AP6.1 Vista en planta del chip 74LS08

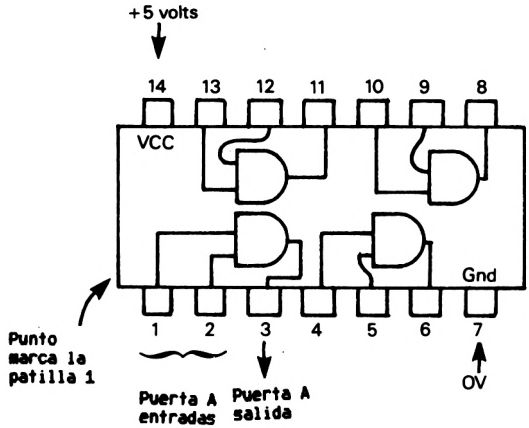
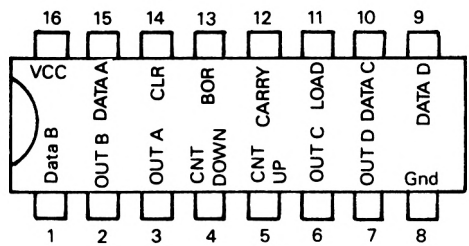


Fig AP6.2 Vista en planta del chip 74LS193

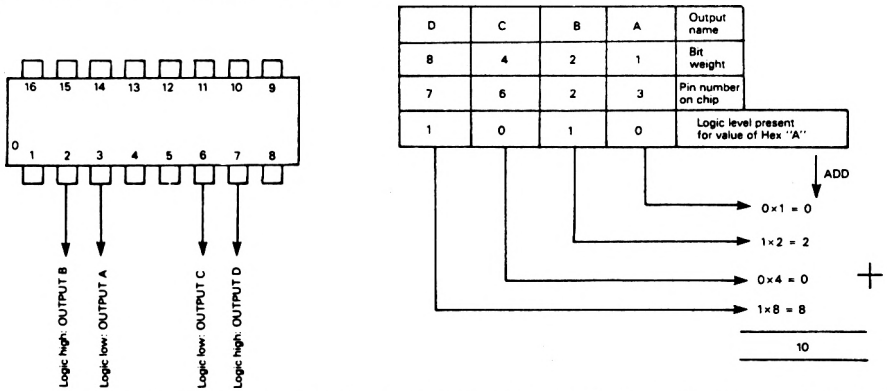


Ahora que hemos visto las puertas, echemos una ojeada a elementos ligeramente más complejos. Los contadores son otros elementos lógicos que son cruciales en el funcionamiento de un ordenador, y en muchas otras aplicaciones de lógica electrónica. Los contadores se pueden usar para muchas cosas. Como ejemplo: contadores de retardo de tiempo, contadores de repetición de secuencia, contadores de impulsos, contadores de sensores de posición, relojes digitales - la lista es interminable. Tomemos otro ejemplo concreto.

El 74LS193 es un chip de la familia TTL, que contiene un contador que tiene la habilidad de contar, en binario, desde cero a quince. Puede contar hacia arriba o hacia abajo, se le puede poner a cero, o puede producir una señal de acarreo para otro contador (de la misma forma que el contador de segundos de su reloj digital que cuando llega a 60 produce una señal de acarreo para suma uno al contador de minutos). La figura AP6.2 muestra una vista en planta de un 74LS193. Como el anterior, es un paquete DIL, pero esta vez con 16 patillas. También como el 74LS08 tiene las mismas patillas en las esquinas para la tensión de alimentación de +5V y la masa, en las patillas 16 y 8 respectivamente. Volvamos ahora al principio de este capítulo donde revisábamos el sistema binario. Las patillas 3,2,6 y 7 son las salidas del contador. Esto es, en estas patillas estarán los niveles lógicos uno y cero que compondrán el valor binario del número que tenga el contador. Si, por ejemplo, fuera como el mostrado en la figura AP6.3, el valor del contador sería diez (o el valor Hex A). Podemos poner fácilmente a cero el valor del contador, aplicando un uno lógico a la entrada CLR en la patilla 14. Solo se debe mantener alto temporalmente, ya que el contador se limpiará continuamente, y no puede contar hasta que aparezca un cero lógico en la entrada CLR. A continuación veremos como hacer contar al 74LS193. Como ya dijimos, puede contar hacia arriba o hacia abajo, esto es, cero, uno, dos, tres, cuatro,..... o quince, catorce, trece, doce,....y así sucesivamente. La forma en que se puede hacer que el chip cuente hacia arriba es aplicar los impulsos de tensión a la patilla 5 mientras que la patilla 4 se conecta a uno lógico. Los impulsos de tensión suelen tomarse de un circuito de reloj. Este circuito no tiene nada que ver con un reloj para contar tiempo y fechas, sino que genera impulsos a intervalos regulares. La mayoría de los circuitos de ordenadores usan impulsos de reloj de una fuente u otra. En el CPC 464, por ejemplo, hay un cristal oscilador que proporciona una frecuencia de reloj de 4 millones de impulsos por segundo. Esta se usa extensivamente en los proyectos detallados en este libro. Así que aplicando los impulsos del reloj a la entrada CNT UP del 74LS193, en la patilla 5, cada impulso que entre por esta patilla incrementará el valor contado en uno. (En informática, el sumar uno a un valor de un contador se denomina incrementar). Para hacer que cuente hacia atrás, debemos mantener la patilla 5 a nivel lógico uno, y aplicar los impulsos del reloj a la patilla 4, la entrada CNT DOWN. Las entradas de contador del 74LS193, y la de la mayoría de los chips contadores TTL y CMOS, son de un tipo especial llamado entrada disparada por transición. Deben ser de un tipo especial, debido a la gran velocidad a la que tiene que trabajar. Para explicarlo, el 74LS193 es capaz de contar 25 millones de impulsos de entrada por segundo (obviamente solo puede contar hasta 15, por lo que el 74LS193 se recicla 156250 veces para 25 millones de impulsos de entrada, 16 veces 156520 = 25,000,000). Para que no se pierdan impulsos o no se cuenten dos veces, el 74LS193 modifica el valor del contador cuando detecta un nivel lógico cero cambiando a uno, en sus entradas de cuenta. A esto se le llama entrada de conta-

dor disparada por transición positiva. Una entrada de transición negativa (como la que se usa en el chip 74LS93) es activada cuando cambia de uno lógico a cero lógico, y se le llama entrada de contador disparada por transición negativa. Otros tipos de chips tienen entradas disparadas por transición positiva.

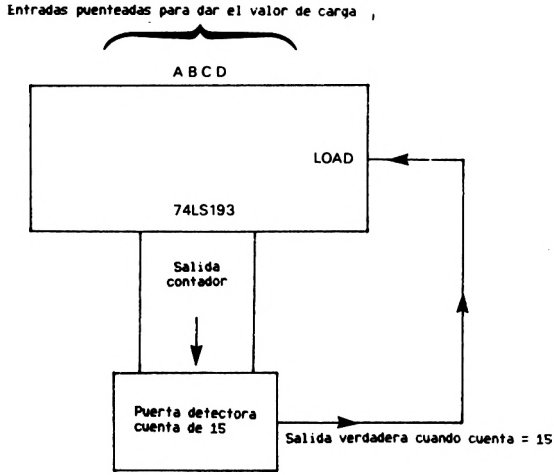
Fig 6.3 Niveles lógicos de salida del chip contador 74LS193, cuando el valor del contador = Hex "A" (10 decimal)



La última facilidad que vamos a ver del 74LS193, es su capacidad de preselección. Mire la patilla 11 en la figura AP6.2. Se llama LOAD. Su función es la de entrada de carga que nos permite colocar en el contador cualquier valor requerido. El modelo binario de ceros y unos que representa el número que quiere poner en el contador, se introduce en las entradas DATA A, DATA B, DATA C y DATA D, que son las patillas 15, 10, 9 y 1 respectivamente. Para cargar el contador se pone a nivel bajo la patilla LOAD, que debe permanecer e nivel alto durante la operación normal del contador. Muchos lectores preguntarán, "Alan, ¿para qué podemos querer preseleccionar el valor del contador?. Hay muchas razones. Pongamos que usted está construyendo un temporizador electrónico que debe contar hacia abajo en segundos. Puede usar fácilmente el chip 74LS193 como la base de ese contador, pero tenga en cuenta lo que ocurre cuando cuenta hacia abajo en segundos. Cuando llegamos a, por ejemplo, treinta, la siguiente cifra a contar será veintinueve. Por lo tanto el dígito menor del contador debe cambiar de cero a nueve. ¿Puede hacer esto el 74LS193 sin ayuda?, no, no puede, su secuencia es 3, 2, 1, 0, 15, 14, 13, etc. Por lo tanto necesitamos un circuito externo para detectar cuando llega el contador a 15 (1111 en binario). La figura AP6.4 muestra un diagrama de bloques del circuito extra que se debe conectar. Este circuito activa la entrada LOAD del contador, que debe ponerse a 9 (1001 en binario) en las entradas DATA. Hay muchos otros usos para la facilidad de preselección y carga del 74LS193, para darle otro ejemplo importante, la división de frecuencias.

Brevemente, el divisor de frecuencia es otra forma de contador que se puede utilizar. El esquema básico es, introducir dentro de un contador una señal de onda cuadrada de frecuencia conocida, y obtener un submúltiplo de esa frecuencia. Por ejemplo, si introduce una frecuencia de 10,000 Hz en su chip 74LS193, la señal de salida en QD, patilla 7, será 10000 dividido por 16 = 625 Hz. Dividimos por 16 porque

Fig AP6.4 Diagrama de bloques del esquema de detección y recarga del contador

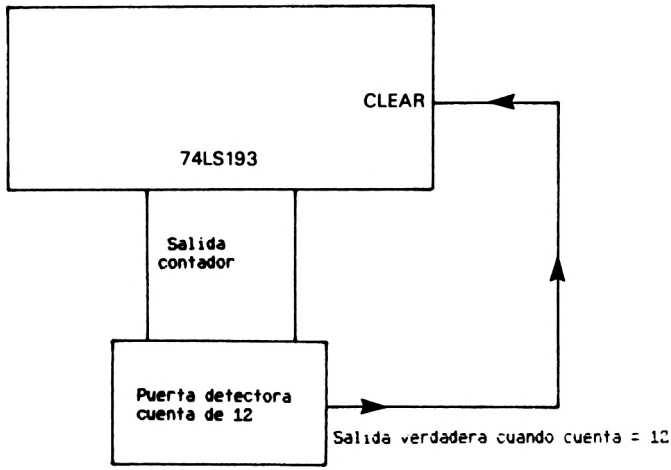


en esta aplicación tenemos que incluir el cero como un estado válido del contador. Podemos usar algunos circuitos lógicos extra para detectar un valor determinado del contador, la salida de esta lógica debe conectarse a la patilla 14 del 74LS193, entrada CLR. A esto se le llama modificar la longitud del contador. Si el valor elegido es 12, siempre que el contador llegue a 12 , los circuitos lógicos externos lo restaurarán a cero. La figura AP6.5 muestra el diagrama de bloques de la conexión de estos circuitos externos. El contador tiene ahora una longitud de doce. Realmente nunca llegará a doce, porque tenemos que incluir el cero. Mediante este truco podemos hacer un divisor por cualquier número entre 1 y n, donde n es el máximo valor que puede contar el contador.

El último chip que vamos a ver es el 74LS138. Quizás llegue a pensar que este es uno de mis chips favoritos, cuando vea los esquemas de los proyectos del capítulo tres. Al chip 74LS138 se le conoce oficialmente como decodificador de 3 a 8 líneas. Veamos lo que significa.

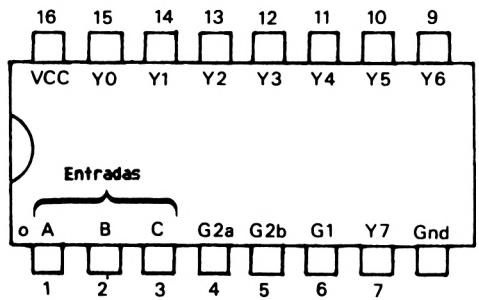
La figura AP6.6 muestra el diagrama de las patillas del chip 74LS138. Como es usual, el chip tiene sus patillas de alimentación en las esquinas, esto es, la patilla 16 es VCC y la patilla 8 es masa. Tiene 8 salidas, llamadas YO-Y7, y varias entradas, de las que A, B y C son las principales. Si ignoramos el resto de las entradas por el momento, será más fácil entender el funcionamiento de este circuito. Usaremos una vez más la representación binaria, el número binario presente en las entradas A, B y C, le dicen al chip cual de las salida Y debe activar. Como las salida Y están normalmente a nivel lógico alto, la salida seleccionada se pone a nivel bajo. Así, si introducimos el valor 5 (101 en binario) en las patillas ABC, se pondrá a nivel bajo la salida Y5 en la patilla 10 del chip. La función del

Fig AP6.5 Uso de una puerta para modificar la longitud del contador 74LS193



del resto de las patillas de entrada del chip, G1, G2a y G2b, es activar o desactivar esta función básica. Con esto queremos decir que si G1 está a nivel bajo, o si G2a o G2b están en nivel alto, la decodificación de las entradas ABC no ocurre. Por lo tanto, para permitir que suceda la decodificación, la entrada G1 debe estar a nivel alto al mismo tiempo que G2a y G2b están a nivel bajo.

Fig AP6.6 Vista en planta del 74LS138



El chip 74LS138 se usa como decodificador de direcciones. Recuerde el principio de este apéndice, cómo las salidas del microprocesador Z80A mandan un número binario al 'bus' de direcciones, para decir la posición a la que quiere acceder. Debe haber algún otro circuito que compruebe constantemente esas líneas de dirección y pasa el impulso de selección al chip direccionado. Veamos un ejemplo, aparte del CPC, para ver cómo se puede usar el chip 74LS138 para decodificar una dirección. Ponga atención, ya que esta vez vamos a usar un esquema real. Digamos que queremos producir una impulso de selección de nivel bajo cuando la dirección que manda el Z80A sea Hex FFF3. A continuación representamos el estado lógico de cada línea cuando selecciona la dirección Hex FFF3:

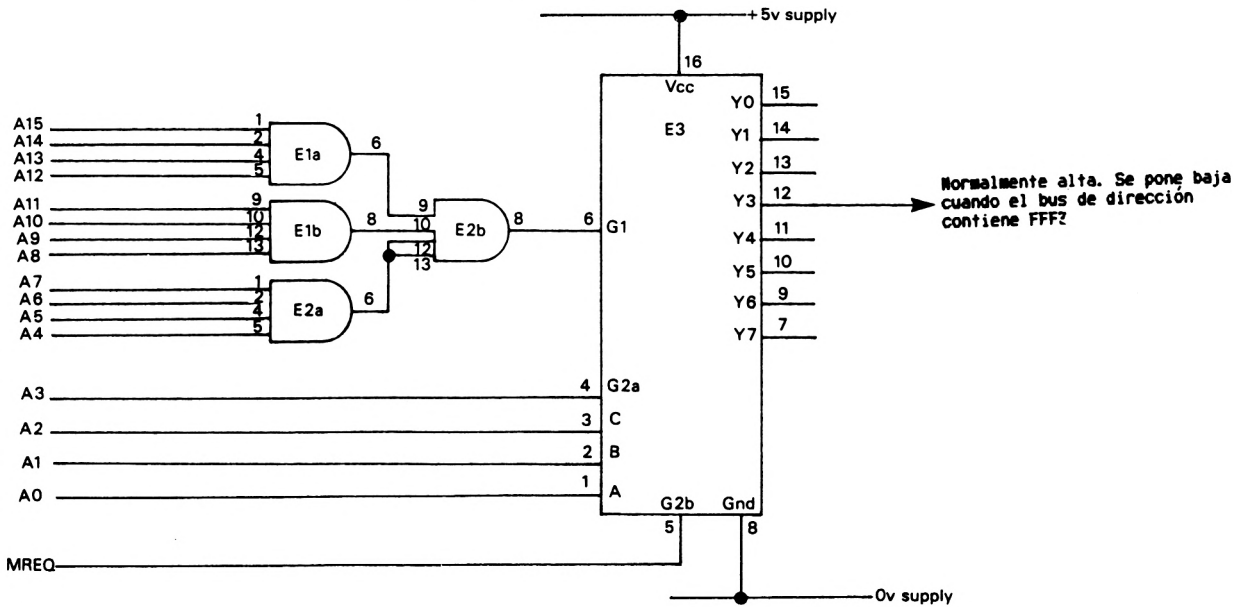
A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
	F				F				F				3		

Ahora usaremos nuestro 74LS138 como elemento central de un circuito decodificador de dirección. El chip decodificador necesitará cierta ayuda de otras puertas lógicas. Mirando el modelo binario anterior, podemos ver que debemos introducir las líneas de dirección A00, A01 y A02 en las entradas A, B y C del chip decodificador, que decodificará la parte de la dirección de valor binario 011 (3 decimal). La línea A03 estará baja cuando esté presente la dirección requerida, así que lo conectaremos a la entrada G2a, que debe estar baja para activar el decodificador. Dos chips de dos puerta AND de cuatro entradas se hacen cargo del resto. Habrá cuatro puertas AND con cuatro entradas cada una. Como ya dijimos, la salida de cada puerta AND se pone en estado lógico alto cuando todas las entradas están en estado alto, así que conectaremos las puertas AND como se ve en la figura AP6.7. Solo necesitaremos tres de las puertas para conectar las líneas A04-A15. La cuarta puerta AND, que tiene su salida conectada a la entrada G1 del decodificador, se usa para detectar cuándo las otras tres puertas AND tienen todas sus entradas a nivel alto.

El resultado de todo esto es que la entrada G1 del 74LS138 recibirá un uno lógico, activando el decodificador cuando las puertas AND detecten que las líneas de dirección A04-A15 están todas a nivel alto. Esto nos deja la línea G2b que conectaremos a la señal llamada MREQ negativa. Esta línea se pone a nivel bajo en el sistema Z80, cuando requiere una lectura o escritura de memoria. Para resumir, las entradas ABC están activadas para decodificar las líneas A00-A02 solamente cuando las puertas AND han detectado que todas sus entradas están a nivel lógico uno, y por lo tanto han introducido un uno lógico en la entrada G1, A03 está a nivel bajo, y la línea MREQ, conectada a la entrada del decodificador G2b, está a nivel bajo. Cuando se cumplan todas estas condiciones, la salida Y3, patilla 12 del decodificador, se pondrá a nivel bajo.

Con esto terminamos nuestra breve revisión de algunos chips reales. Estoy seguro que, para muchos lectores, la parte anterior de este apéndice habrá sido bastante dura, pero estos conceptos se usarán una y otra vez, así que estará bien empleado el esfuerzo invertido en comprenderlo perfectamente. Para introducirse en los aspectos más sofisticados de la lógica electrónica como unidades aritméticas, generadores de paridad, conversores de código, etc, recomiendo a los lec-

Fig AP6.7 Circuito decodificador de la direccion Hex FFF3 - ver texto

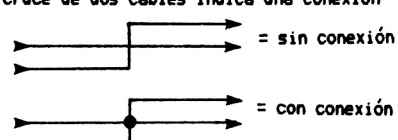


Chips usados

E1 & E2 son 74LS21 - Dos puertas AND de cuatro entradas

E3 es 74LS138 - decodificador 3 a 8

Nota: en los diagramas, un punto en el cruce de dos cables indica una conexión



tores que se hagan con un libro como el "Data Book" de Texas Instruments, o el "CMOS Data Book" de RCA. No son libros baratos, pero le proporcionarán una fuente interminable de información, no solo sobre el rango de chips disponibles, sino también sobre su uso. Tenga en cuenta que una gran cantidad de los diseños de circuitos que puede encontrar en las revistas de electrónica, están basados en los ejemplos de aplicación que se encuentran en este tipo de libros.

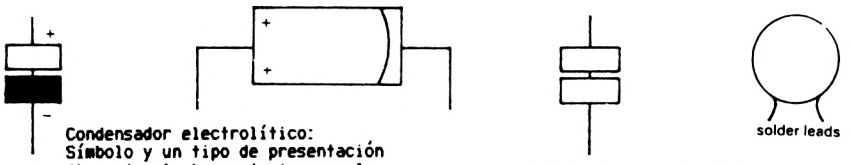
COMPONENTES DISCRETOS

Un componente discreto es uno que no forma parte de un circuito integrado. Condensadores, resistencias, diodos y transistores, son todos componentes discretos. Con un poco de práctica podrá comprender enseguida los circuitos lógicos, e incluso diseñar los suyos propios. También deberá conocer cómo usar los componentes discretos, así que vamos a dar una breve descripción de las propiedades de cada uno de ellos.

Condensadores

Hay varios tipos de condensadores. Los dos tipos principales son los llamados electrolíticos y los no electrolíticos. Sus símbolos se muestran en la figura AP6.8. La capacidad de un condensador se expresa en micro faradios, y, para los más pequeños, en submúltiplos de micro faradios, llamados nano faradios y pico faradios. 1000 pico faradios = 1 nano faradio. 1000 nano faradios = 1 micro faradio. La figura AP6.9 muestra las notaciones usuales.

Fig AP6.8



Condensador electrolítico: Símbolo y un tipo de presentación (hay otros). La parte + se suele marcar con rojo para identificarla

Condensador no electrolítico: Símbolo y un tipo de presentación

Condensadores: Asegúrese de que la tensión marcada en el condensador es mayor que la tensión que intenta aplicarle.

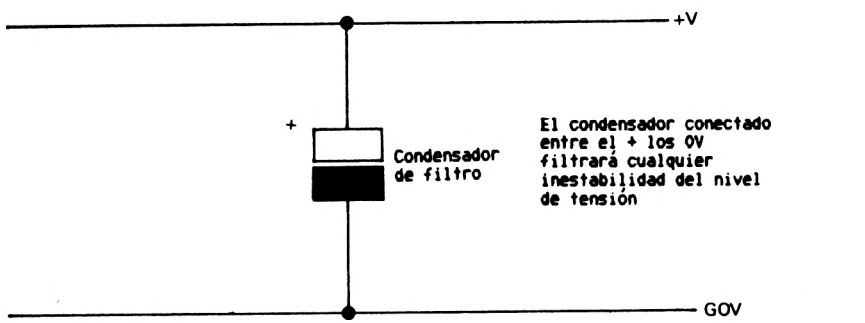
Fig AP6.9 Símbolos de capacidad.

F = símbolo del Microfaradio
nf = símbolo del nanofaradio
pf = símbolo de picofaradio

1000 pf = un nanofaradio
1000 nf = un microfaradio

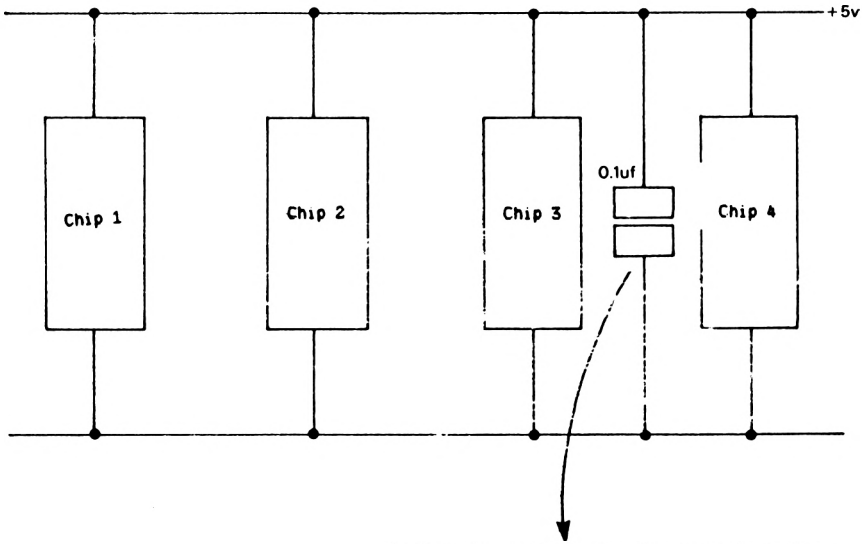
Hay dos propiedades de los condensadores que se suelen usarse en los ordenadores. La primera es que un condensador puede retener una carga eléctrica, no muy grande para el tamaño de condensadores que vamos a usar, pero suficientemente grande como para usarlos como estabilizadores en señales inestables. Cuando los usamos de esta forma, los conectamos como se ve en la figura AP6.10. Otro uso de los condensadores es hacerlos que se descarguen lentamente a través de una resistencia, usando un chip para detectar la altura de la carga que queda. Esta idea forma las bases de los circuitos temporizadores, ya que los condensadores suelen tardar el mismo tiempo para llegar a un nivel dado de carga en cada operación. Uno de los usos más comunes en ordenadores es como desacopladores. La acción de un chip TTL en conmutación puede causar picos indeseables en la línea de alimentación, afectando a otros chips. La figura AP6.11 muestra cómo se conecta, dando una capacidad local a la línea de +5V en el punto en que se necesita.

Fig AP6.10 Condensador conectado como dispositivo estabilizador



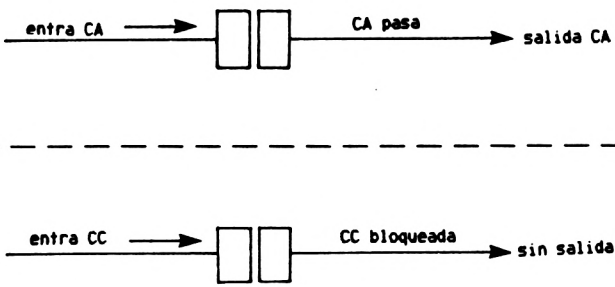
Un condensador usado de esta forma recibe el nombre de condensador de reserva.

Fig AP6.11 Condensador de desacoplo



La otra forma de usar un condensador es como bloqueo para corriente contínua. Cuando conectamos un condensador, como se ve en la figura AP6.12, la señal alterna pasará a través de él, mientras que bloqueará la corriente contínua. Esta propiedad se usa ampliamente en circuitos de audio.

Fig AP6.12 Diagrama de cómo bloquea la corriente continua un condensador.



Resistencias

En la figura AP6.13 se muestran los símbolos de las resistencias de valor fijo y variable. Se usan cuando se quieren producir señales electrónicas, para contener su circulación por los circuitos. La función de una resistencia en el circuito de un ordenador es, principalmente, limitar la corriente que circula por sus componentes, como los transistores. Si no usáramos resistencias para limitar la circulación de corriente en ciertas partes del ordenador, el flujo de corriente a través de ellos podría dañarlos. Piense en su propio televisor, subiendo el volumen lo más alto que podamos. ¿Puede notar la distorsión del sonido? Esto se debe a que ha eliminado la resistencia a la señal dentro del amplificador de audio, y ahora está sobrecargado. Vuelva a bajarlo rápidamente antes de que se dañe algo. Ahora puede oírlo, pero puede pasarle algo similar en circuitos que manejen luces o relés. Cuando se sobrecargan se ponen muy calientes y pueden dejar de funcionar. Por lo tanto, las resistencias se usan para prevenir las sobrecargas. Pero también se pueden usar, junto con los condensadores, en circuitos temporizadores, y también para forzar entradas lógicas a uno u otro nivel, y para muchas otras funciones. Las resistencias se miden en Ohmios, y cada resistencia tiene un código de colores que indica cuántos Ohmios de resistencia tiene. La figura AP6.14 muestra este código. Las resistencias se suministran en varias potencias distintas. 1/4 de watio, 1/2 watio, 1 watio y 10 watios, son las más comunes. Debe usar siempre las potencias especificadas en las listas de componentes de los proyectos.

Fig AP6.13 Símbolos de las resistencias variables y fijas

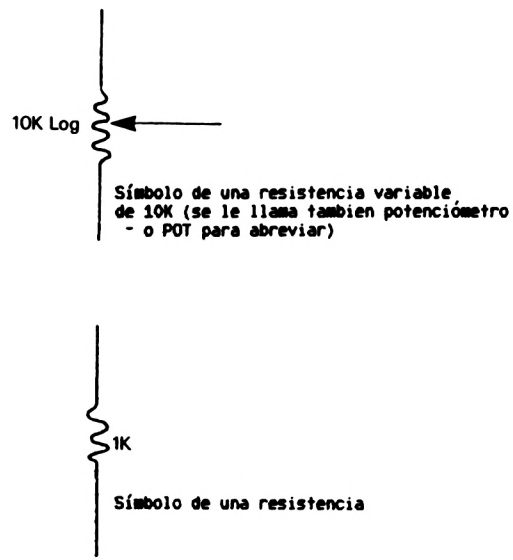
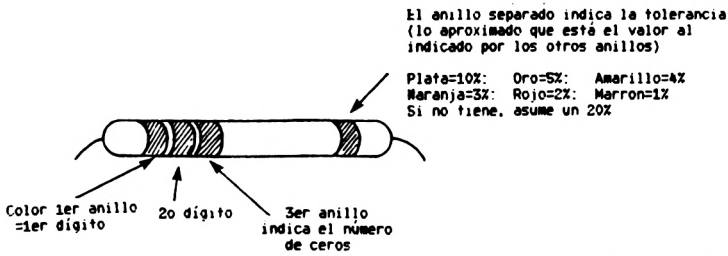


Fig AP6.14 Código de colores de las resistencias



Color	No.
Negro =	0
Marrón =	1
Rojo =	2
Naranja =	3
Amarillo =	4
Verde =	5
Azul =	6
Violeta =	7
Gris =	8
Blanco =	9

Diodos

En la figura AP6.15 puede ver el esquema de un diodo. La propiedad de un diodo es que la electricidad pasa en el sentido que marca la flecha, pero no en sentido contrario. Así forma una calle de sentido único para la electricidad. Es muy útil cuando conecta cuatro diodos para formar un puente rectificador. La tensión alterna se introduce por los terminales marcado AC, y en los terminales marcados + y - obtiene una tensión continua (DC) que será filtrada por un condensador de gran capacidad, puede usarse como entrada para un regulador de tensión. (ver capítulo tres). En ordenadores, los diodos se pueden usar para formar puertas OR, como se ve en la figura AP6.16. Si la entrada de cualquiera de los diodos se pone alta, el otro extremo unido de todos los diodos, se pondrá también a nivel alto. Existe un tipo especial de diodos llamados Zener. Estos diodos, cuando se conectan en una configuración como la que se muestra en la figura

AP6.17 proporciona una tensión de referencia. La tensión que se obtiene debe ser especificada por usted cuando compra en diodo zener. El que se ve en la figura AP6.17 es un diodo zener de 3.5 voltios, por lo tanto la tensión de referencia obtenida será de 3.5V. Tenga cuidado de no sobrecargar el diodo zener, poniendo un valor de R demasiado pequeño, y el valor de V debe ser siempre por lo menos un voltio mayor que la tensión que desea obtener como referencia del diodo zener. Otro tipo de diodo es el LED (Ligth Emiting Diode - Diodo emisor de luz). Cuando pasa la corriente a través de este tipo de diodo, emite luz. Los LED necesitan resistencias conectadas en serie, como se ve en la figura AP6.18, para limitar la corriente que circula a través de ellos.

Fig AP6.15 Tipos principales de diodos.

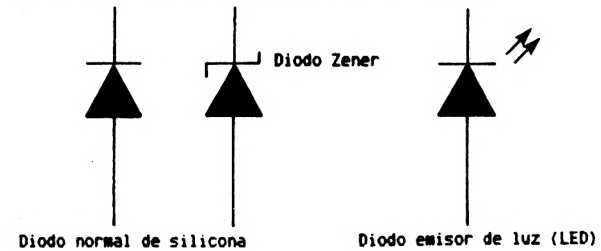
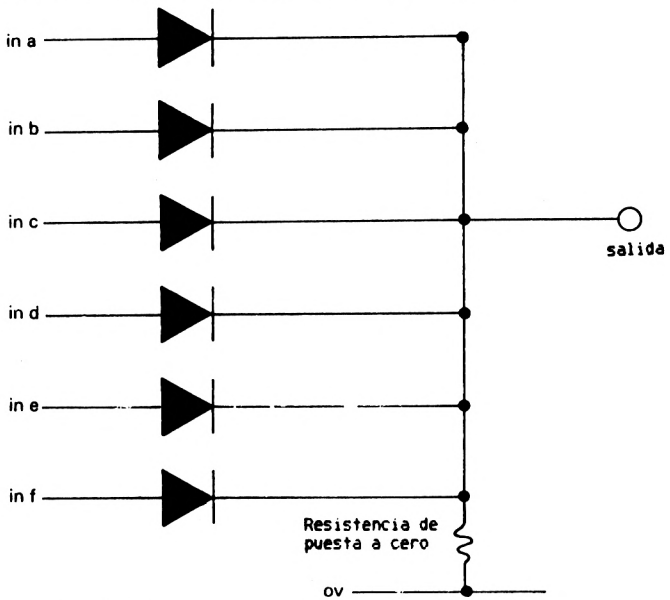


Fig AP6.16 Puerta OR construida con diodos



Nota: Este tipo de puerta tiene menor rendimiento y calidad que un chip equivalente. Poniendo alta cualquier entrada, se consigue una salida alta.

Fig AP6.17 Circuito con diodo zener.

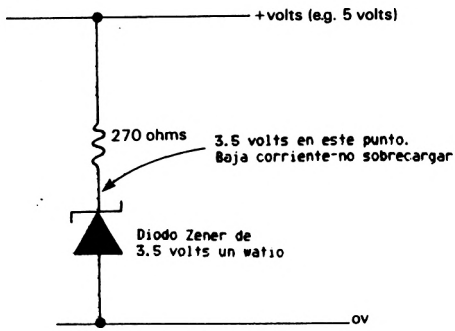
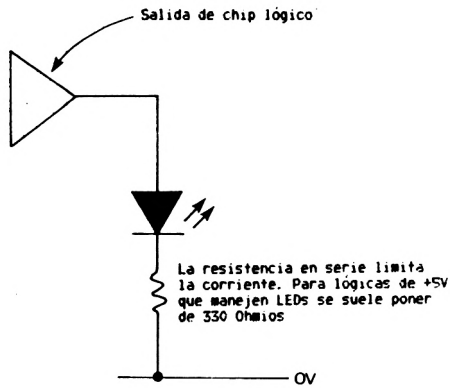


Fig AP6.18 Conexión típica de un LED



Transistores

Estos dispositivos es con los que están hechos los modernos circuitos electrónicos. Los chips contienen cientos de transistores integrados, y también se usan transistores discretos (esto es, uno en cada paquete) para conectar señales lógicas relativamente débiles a LEDs, relés, o motores. En los ordenadores, los transistores se usan como conmutadores controlados por señales lógicas. Refiérase a la figura AP6.19 - muestra los símbolos de los dos tipos básicos de transistores. El transistor NPN conmuta cuando tiene un uno lógico en la base. El transistor PNP conmuta cuando su base se pone a nivel bajo. Esta descripción nos sirve para usar los transistores como conmutadores controlados por señales lógicas, pero tenga cuidado para otros usos, porque debe conocer más detalles sobre su operación. Cuando compre un transistor, intente conseguir la hoja de características. Le costará muy poco más, pero le dirá todo lo que tiene que saber de él. Una cosa que sí debe conocer es qué patillas corresponden a la base, al emisor y al colector. Muchos catálogos de los principales distribuidores de componentes electrónicos incluyen información de la distribución de las patillas. Los transistores discretos suelen ser sensibles al calor cuando se les suelda, así que tenga cuidado. La figu-

ra AP6.20 muestra cómo puede conectar un transistor NPN como inversor. Observe la resistencia conectada a la base, para limitar el flujo de corriente.

Fig AP6.19 Símbolos de los transistores.

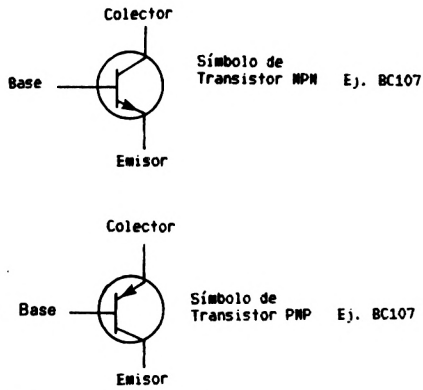
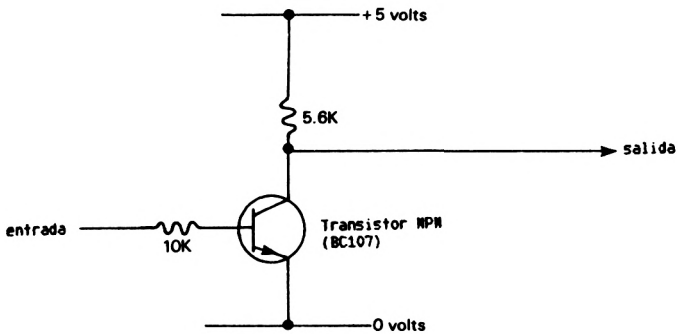


Fig AP6.20 Uso de un transistor como inversor



Una entrada alta hace que el transistor se active, esto hace que la tensión del colector caiga a menos de un voltio. Cuando la entrada se pone baja, el transistor se desactiva, y la resistencia de 5.6K pone la salida del colector alta. La resistencia de 10K limita la corriente que pasa a la base del transistor.

EL PROCESO FÍSICO DE CONSTRUCCIÓN

Los métodos usados para construir los proyectos detallados en este libro se describen fácilmente. Esto se debe a la disponibilidad de los circuitos impresos (PCB). El diseñador de un circuito impreso crea pequeñas pistas de cobre en la placa, para hacer todas las conexiones necesarias entre los componentes. Todo lo que necesita usted es soldar cada componente en su posición, y hacer después las conexiones externas. Una vez comprobado un par de veces, que no hay gotas de soldadura en la placa y que todos los componentes se han insertado de forma correcta, el trabajo está casi terminado. Si usted diseña sus propios proyectos, tendrá que construirse el circuito impreso o usar algún tipo de placa preparada con pistas paralelas, que venden en cualquier comercio de componentes electrónicos. Este último sistema es más lento que el de construirlo sobre el circuito impreso, ya que debe hacer todas las conexiones con trozos de cable y debe tener mucho cuidado de no confundirse al hacer las conexiones. No tengo intención de dar más detalles sobre la construcción en este tipo de placas, excepto recomendarle que para la construcción de proyectos para ordenadores, use una de las varias placas para micros disponibles en el mercado.

Soldaduras

Soldar es fácil, pero por desgracia es muy fácil hacerlo mal. La regla de oro cuando se suelda es calentar la soldadura, no el estaño. Las figuras AP6.21 y AP6.22 muestran lo que quiero decir. Debe calentar el punto a soldar y usar ese calor para fundir el estaño. NO DEBE calentar el estaño y dejar que caiga la gota fundida sobre el contacto que debe soldar, porque producirá soldaduras frías que, cuando empiecen a funcionar, le causarán problemas. La soldadura perfecta requiere un soldador con una punta del tamaño adecuado, y un paño o esponja con la que limpiar la punta cuando se ensucie. No puede hacer buenas soldaduras con un soldador sucio. No es necesario decir que el soldador debe ser bueno y estar suficientemente caliente, así que el viejo tipo de soldador que se calentaba sobre el fuego se debe desechar definitivamente.

Fig AP6.21 Soldadura mala.

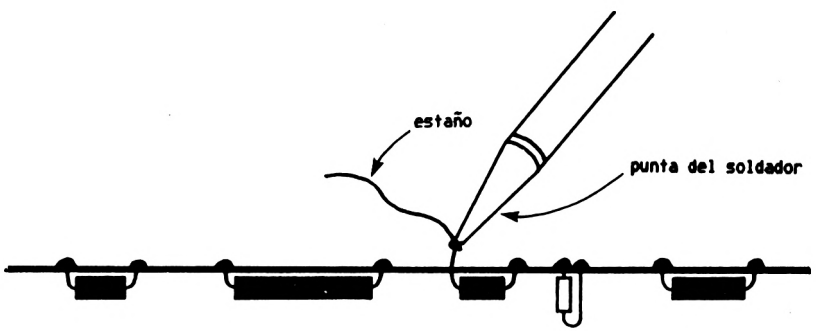
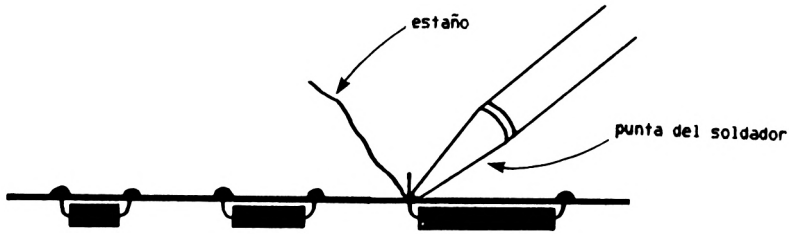


Fig AP6.22 Soldadura buena.



Peligros

Con este tema podríamos llenar un capítulo entero. Los principales peligros en la construcción de circuitos son realmente evitables con un poco de cuidado. Aquí tenemos una lista de las cosas que se deben hacer para evitarlos:

1) Use siempre zócalos, son baratos y los soldará en lugar de los circuitos integrados, evitando la posibilidad de dañar los chips con el calor. Asegúrese de que aplica el calor del soldador a los componentes durante el mínimo tiempo posible. Los cristales están especialmente expuestos a dañarse por el calor.

2) Haga una inspección visual después de cada soldadura, para asegurarse que no ha hecho un puente con otra conexión adyacente. Este tipo de puentes pueden hacer que no funcione adecuadamente el circuito.

3) Cuando diseñe sus propios circuitos, use componentes que no tengan pequeñas capacidades espúreas. Esta aplican capacidades extras en los transistores de potencia, y también en los circuitos lógicos. Cuando haya construido el circuito, puede darse cuenta de que no funciona como esperaba.

4) Compruebe siempre un par de veces que no hay cortocircuitos entre las líneas de +12V y los -12V, y otras partes del circuito. Un cortocircuito desde los +5V con otra parte del circuito no producirá ningún daño, pero un cortocircuito entre las líneas de 12V siempre suele dañar algún componente, y lo peor es que puede llegar a dañar su CPC.

5) Incluya siempre todos los componentes que se muestran en los esquemas, a menos que se especifique que son opcionales. Puede parecerle correcto eliminar un condensador de desacoplo o un diodo, pero el diseñador los puso por alguna razón. Si omite componentes, sufrirá la eficacia del circuito.

6) Cuando construya un kit, asegúrese de que ha comprado todos los componentes. Los fabricantes de kits suelen verse forzados algunas veces a proporcionar dispositivos equivalentes a los que se muestran en los diseños. (Suelen hacerlo con permiso del diseñador, por lo que no debe tener problemas de compatibilidad). Si es este su caso, asegúrese de que conoce cuales han sustituido, y de que no mezclará los componentes cuando los inserte

en el circuito. Es fácil de hacer, yo lo hice. Otra cosa a tener en cuenta en la construcción de kits es asegurarse de leer cuidadosamente los detalles de construcción. Deben venir con el kit, o deberían venir con el proyecto que está construyendo. Si hay algún puente que cablear, asegúrese de insertarlo correctamente.

7) Conecte siempre la masa (o línea de cero voltios de la fuente de alimentación) cuando el circuito está desconectado. El tener +12V y -12V conectados sin masa hace que tengamos 24 voltios reales en el circuito. Se puede recalentar y dañar muy rápidamente, y puede incluso dañar el CPC.

8) Desconecte siempre su circuito del CPC cuando lo haya apagado. Solamente debe conectarlos al CPC cuando los haya comprobado y esté seguro de no tener cortocircuitos.

Esto concluye la lista de cosas que debe hacer para evitar los peligros, la lista de lo que no debe hacer es la opuesta a la que acabamos de darle. Lo único que queda por decir que no debe hacer, aunque es bastante obvio, mucha gente lo hace, no conecte a los circuitos lógicos ninguna otra tensión diferente a las especificadas. Si lo hace conseguirá tener un CPC y un proyecto humeante, así que no haga tonterías.

FUENTES DE ALIMENTACIÓN

Vamos a tratar ahora varios puntos generales sobre las fuentes de alimentación. Primero, respecto a las tensiones positivas y negativas. Todos los niveles de tensión marcados en una fuente de alimentación se expresan en relación con su masa o salida de cero voltios. Los términos cero voltios y masa los usa mucha gente para describir lo mismo cuando hablan de fuentes de alimentación. El término tierra se refiere siempre a la conexión a tierra del enchufe, que debe conectarse siempre al chasis de la fuente de alimentación o a la caja. Esto le asegurará que nunca entrará en contacto con la caja el polo activo de la corriente alterna, el fusible saltará inmediatamente. (Si no se conecta la caja a tierra y sucede lo que hemos dicho, la primera persona que toque la caja puede quedarse electrocutada). Por lo tanto, la salida de cero voltios de la fuente de alimentación es la que se toma como referencia para las otras salidas. Algunas veces es muy difícil para los principiantes en electrónica comprender las tensiones negativas. Pongamos una analogía no electrónica para clarificar este concepto.

Imagine un ventilador bidireccional, de los que hay en alguna oficinas. Imagine que está usted sentado a su lado. Está apagado, así que, ¿qué tipo de presión de aire experimenta usted?, ninguna. A este estado le llamaremos de no presión. Imagine ahora que alguien enciende el ventilador de forma que saque el aire de la oficina a la calle. Ahora puede sentir presión, es una presión negativa. ¿Negativa compa-

rada con qué?, comparada con el estado de no presión que existía anteriormente. Ahora esa misma persona invierte la acción del ventilador. Ahora está soplando aire desde la calle hacia la oficina. Lo que usted siente ahora es una presión positiva. Podemos entender estos conceptos, pero no son realmente muy diferentes de los de tensión positiva y negativa de una fuente de alimentación de un ordenador. Los cero voltios son como el estado de no presión, la tensión negativa es como el estado de presión negativa, y la tensión positiva es como el estado de presión positiva. Pensando de esta forma en las salidas de la fuente de alimentación, nos puede ayudar a visualizar lo que son exactamente.

Las conexiones entre la fuente de alimentación y los circuitos son muy importantes. Deben ser sólidas, sin peligro de roturas. Se deben hacer con cables apropiados para una corriente, por lo menos, la mitad superior a la máxima que deben conducir. Por último, deben ser lo más cortas posible, para evitar que capten ruidos. Este tipo de líneas son muy sensibles a estos problemas si son demasiado largas.

TRASMISIÓN DE DATOS EN SERIE

Como punto final de este pequeño curso de 'hardware', echemos una ojeada a algo tan importante como la trasmisión de datos en serie.

La trasmisión de caracteres a un VDU (o a una impresora seria) involucra la trasmisión desde el ordenador a la VDU de los códigos binarios de cada carácter que debe aparecer en la pantalla. El código binario de cada carácter se toma de uno de los varios juegos de caracteres estándar que se usan en la industria de los ordenadores. El más extendido es el llamado ASCII. Es la abreviatura de American Standar Code Information Interchange (Código americano estándar para intercambio de información). Este juego de caracteres es el que usa el Amstrad. Por ejemplo, el código del número cero (0) es 48 decimal. (Pruebe usted mismo tecleando "PRINT CHR$(48)" desde el BASIC. Imprimirá el carácter cero). El juego de caracteres ASCII lo puede encontrar en el apéndice cuatro de este libro.

El juego de códigos ASCII usa siete bits, pero los caracteres trasmitidos al VDU suelen contener ocho bits, el octavo bit se suele poner a cero, o se usa para otros propósitos especiales que no nos interesan ahora.

En los grandes ordenadores, que son capaces de proporcionar facilidades a muchos usuarios simultáneamente, el método más usado para conectar una VDU (Unidad de visualización) al ordenador es por medio de una línea serie. Con este método, la VDU se puede situar a cualquier distancia del ordenador y, usando un modem y una línea telefónica, el VDU puede estar al otro lado del mundo.

Para ver un ejemplo más normal, imaginemos una factoría que tiene un departamento de informática en un lado de un gran edificio. El director de la compañía decide que le gustaría tener una VDU sobre su mesa, para poder

ver los últimos datos de ventas, detalles de la nómina etc. Por desgracia su oficina está en la parte opuesta del edificio. ¿Cómo conectaremos la VDU al ordenador?, ¿Usaremos niveles lógicos estándar y

mandaremos las señales por ocho cables desde el ordenador a la oficina?. Lo podríamos hacer, pero hay varias objeciones. Una factoría es un lugar susceptible de que haya gran número de máquinas eléctricas, tornos, conductores, taladradoras, etc. Esto produce multitud de interferencias eléctricas - del tipo de las que nos estropean los programas de televisión. Si nuestro cable de ocho conductores pasara cerca de de estos, el ruido eléctrico al que están sujetos podría cambiar fácilmente el estado lógico de uno de los bits, y los datos que llegarían al VDU estarían corrompidos. La otra objeción, menos técnica pero igualmente relevante, es el costo del cable de ocho conductores.

Hace tiempo que se encontró poco práctica la idea de trasmitir datos a niveles TTL, sobre cables largos. Esto condujo la desarrollo de un método de trasmisión de datos sobre tres únicos cables, y a niveles de tensión que fueran menos afectados por las interferencias eléctricas. El más conocido de estos métodos estándar es el llamado RS232.

Para trasmitir datos sobre tres cables, desde un ordenador a una VDU, se usan de la siguiente forma:

Un cable como conexión de masa común (o cero voltios).
Un cable para mandar los datos serie desde el ordenador a la VDU.
Un cable para mandar los datos serie desde la VDU al ordenador.

La otra facilidad del RS232 es que los niveles lógicos viajan a través de la conexión serie como niveles de tensión positivos y negativos. Este cambio de niveles se realiza mediante un chip conversor, que convierte los niveles TTL estándar en impulsos de tensión positivos y negativos, preparados para viajar sobre la conexión de tres cables. Concretamente, los niveles lógicos altos viajan como impulsos negativos, mientras que los niveles lógicos bajos lo hacen como impulsos positivos. Las tensiones positivas y negativas deben estar equilibradas, lo que significa usar -12V para el nivel negativo y +12V para el positivo. El rango permitido para la tensión positiva es de +3 a +15V, mientras que para la tensión negativa es de -2 a -15V.

Ahora que hemos establecido los niveles de tensión y que vamos a usar tres cables para la conexión serie, debemos ver la temporización de la trasmisión y recepción de datos.

Hay varios chips diferentes que efectúan la trasmisión de datos en serie. El más simple y, hasta hace poco, más usado de estos chips se llama UART. (¿Más iniciales? - UART es la abreviatura de Universal Asinchronous Receiver Transmitter o Trasmisor Receptor Asíncrono Universal). El chip UART contiene todos los circuitos requeridos para convertir datos en octetos de ocho bits en un flujo de datos serie, y también puede convertir los datos serie recibidos en octetos de ocho bits, para mandarlos a un ordenador. El único circuito extra requerido para construir una unidad completa de trasmisión serie es el de los chips de cambio de nivel mencionados anteriormente, y un chip llamado Baud Rate Generator (Generador de velocidades de trasmisión).

VELOCIDADES DE TRASMISIÓN Y TEMPORIZACIÓN EN TRASMISIÓN SERIE

Como probablemente sabrá, los cristales de cuarzo proporcionan impulsos de temporización bastante exactos cuando se les aplica una tensión. Esto ha hecho que se usen en elementos de temporización en relojes digitales. También se usan ampliamente en circuitos de ordenador que requieren una señal con una frecuencia conocida y exacta. La trasmisión serie requiere este tipo de exactitud. El problema reside en que a cada lado de la conexión serie, esto es en el lado del ordenador y en el de la VDU o impresora, haya una frecuencia idéntica disponible desde un chip generador de velocidad de trasmisión. Estos chips usan siempre un cristal para proporcionar la frecuencia maestra que ellos dividen para obtener una de las dieciséis velocidades estándar en la industria. (Como regla general se puede decir que un BAUDIO es un impulso por segundo - o un Hertz). El UART de cada extremo de la conexión usa una frecuencia para determinar cuándo el nivel lógico de cada uno de los ocho bits de un octeto, está en la línea serie. También deben conocer ambos UART el instante exacto en que empieza la trasmisión del octeto. Usemos el CPC para ilustrar esto. La figura AP6.23 es el listado de un programa BASIC que ilustra gráficamente una trasmisión serie. El concepto de trasmisión se ve perfectamente en este programa, pero hay dos diferencias importantes entre este ejemplo y una trasmisión real. Primero, la trasmisión real es en dos sentidos - hay dos líneas de datos interconectadas. Segundo, en la trasmisión real se trasmiten algunos bits extra, además de los ocho bits de datos.

Algunos de estos bits extra se usan para permitir al dispositivo receptor comprobar que los datos recibidos son correctos. Otro bit extra es el bit de comienzo. ¿Recuerda que dijimos que los UART a ambos extremos de la línea deben saber el instante exacto en que comienza la trasmisión? Pues el bit de comienzo se manda siempre como 0 lógico, mientras que una línea inactiva está siempre a nivel 1. Después de un periodo en el que la línea mantiene el bit de comienzo, el UART receptor detecta el principio de la trasmisión cambiando la línea a 0 lógico. Esto hace que arranque la temporización de la que depende la trasferencia correcta. Los bits extra que quedan se llaman bits de parada, son siempre niveles 0 lógico, y puede tener uno o dos bits de parada.

Como el uso del RS232 se usa en todo tipo de ordenadores de cualquier tamaño, proporciona una forma barata de intercambiar datos entre máquinas. La conexión de tres hilos entre la máquina que manda y la que recibe, es todo lo que se requiere. Este método de trasferencia se suele usar para evitar tener que escribir programas elaborados, para manejar interfaces de 'hardware' diseñados para el usuario.

Hay algunos otros puntos que conocer sobre la trasmisión de datos serie, pero los más importantes se han cubierto en la descripción de los proyectos de RS232 en el capítulo tres.

```
100   '  ** Programa para ilustrar los conceptos de trasmisión serie

200   MODE 2: LOCATE 1,5: PRINT "Teclee un carácter alfanumérico ";:
      WHILE TR$="": TR$=INKEY$: WEND: IF ASC(TR$) > 32 AND ASC(TR$) <
      127 THEN 300 ELSE PRINT CHR$(7): TR$="": GOTO 200
300   PRINT TR$: BIN2$=BIN$(ASC(TR$)): IF LEN(BIN2$) < 8
      THEN BIN2$=STRING$(8-LEN(BIN2$),48)+BIN2$
400   FOR I=1 TO 8: BINARY$=BINARY$+MID$(BIN2$,I,1)+CHR$(32): NEXT I
410   PRINT: PRINT "Demostración Lenta o Rápida? <lenta>": WHILE R$="":
      R$=INKEY$: WEND: IF UPPER$(R$) <> "R" THEN SPD=50 ELSE SPD=10
420   ON BREAK GOTO 32767
500   MODE 1: PRINT "Demostración de trasmisión de datos en serie"
600   MOVE 0,317: GOSUB 900: MOVE 325,15: GOSUB 900: INK 1,26: INK 2,9:
      MOVE 500,200: DRAWR 40,0,1: MOVER -40,-30: DRAWR 40,0,2 '
      ** Dibuja los registros de trasmisión y recepción, prepara las
      ** tintas y después
610   LOCATE 21,13: PRINT "Cero lógico=": LOCATE 21,15:
      PRINT "Uno lógico=" 'imprime en la pantalla una explicación de
      cada estado de la línea de interconexión
700   LOCATE 2,5: PRINT CHR$(34);TR$;CHR$(34);"=ASCII HEX ";
      HEX$(ASC(TR$)): LOCATE 2,3: PRINT BINARY$
750   IK%=1: GOSUB 21000 ' ** Dibuja la conexión serie con INK 1
760   TOSEND=8: EVERY SPD,0 GOSUB 22000
770   GOTO 770 '** esperar hasta que entre en funcionamiento EVERY **

900   ' ** Subrutina para dibujar un recuadro que representa el registro
          de trasmisión serie. La parte inferior izquierda del registro
          será la posición del cursor gráfico. **
1000  DRAWR 275,0,1: DRAWR 0,55,1: DRAWR -275,0,1: DRAWR 0,-55,1:
      DRAWR 0,23,1: DRAWR 275,0,1: RETURN

21000 ' ** Subrutina para dibujar la conexión en el color
          de tinta indicado por la variable IK%
21010 MOVE 323,50: DRAWR -23,0,IK%: DRAWR 0,310,IK%:
      DRAWR -23,0,IK%: RETURN

22000 ' ** Subrutina para que avance un bit la trasmisión
          serie representada en la pantalla
22010 TOSEND=TOSEND-1: LOCATE 2,3: PRINT SPACE$(16-(TOSEND*2));
      LEFT$(BIANRY$,TOSEND*2):
      IK%=1+VAL(LEFT$(RIGTH$(BINARY$,16-(TOSEND*2)),2)): GOSUB 21000
22020 FOR I=5 TO 21: LOCATE 20,I-1: PRINT " ": LOCATE 20,I:
      PRINT MID$(BINARY$,(TOSEND*2)+1,1): FOR P=1 TO SPD: NEXT:
      NEXT: LOCATE 20,21: PRINT CHR$(32)
22050 LOCATE 22,22: PRINT RIGTH$(BINARY$,16-(TOSEND*2)):
      IF TOSEND <> 0 THEN RETURN
22060 LOCATE 22,24: PRINT CHR$(34);TR$;CHR$(34);"=ASCII HEX ";
      HEX$(ASC(TR$)): LOCATE 2,5: PRINT STRING$(16,32): D=REMAIN(0):
      LOCATE 1,12: PRINT "Pulse una tecla": WHILE INKEY$="": WEND: RUN

32767 MODE 2: END
```

Fig AP6.23 Programa de visualización de trasmisión serie

CONCLUSIÓN

Me da la dolorosa impresión de que este capítulo no ha servido de mucho. En tan poco espacio no he podido enseñarlo todo acerca de la electrónica digital, sin mencionar los otros temas . Lo que espero que haya aprendido leyendo este capítulo son los primeros conceptos. También espero que se haya convencido de que el 'hardware' es un tema en el que se puede profundizar. El lado 'software' de los ordenadores está sobrecargado, mientras que el lado 'hardware' tiende a ser olvidado. Hay mucha gente que entiende de 'hardware', pero no hay muchos en el mercado de los ordenadores personales que publiquen libros o vendan productos sobre este tema. Si su primer amor es el 'software', será capaz de escribir buenos programas como resultado de una ampliación de sus conocimientos de 'hardware'. Y, finalmente, puede ahorrar mucho dinero construyendo y diseñando su propio 'hardware'.

¡Espero que siga adelante!

APENDICE SIETE
La Puerta de Impresora

Las conexiones de la puerta de impresora del Amstrad CPC resultan algo difíciles si intenta conectar un cable de cinta estándar. Las referencias en el apéndice apropiado del manual del usuario, muestran que en lugar de una distribución normal de los números de las patillas, se ha usado una distribución inusual. El efecto es que el cable de cinta que conecta usted a la puerta de impresora debe tener conectado en cada contacto un cable distinto, en lugar de la distribución más usual - como la usada en el 'bus' de expansión - donde los números de los conductores van secuencialmente . La figura AP7.1 muestra la distribución de las conexiones de la puerta de impresora.

La puerta de impresora tiene siete bit. El octavo bit está siempre puesto a nivel cero en los datos recibidos por la impresora.

Otro problema relacionado con la conexión de una impresora al Amstrad, y también a otros ordenadores, son las secuencias de escape.

Una secuencia de escape es una facilidad de muchas impresoras modernas. Las secuencias de escape proporcionan una forma de controlar ciertas características especiales de la impresora desde un programa que se está ejecutando. La idea global es usar el carácter de escape ASCII. Las impresoras que facilitan secuencias de escape , consideran el carácter de escape como el primer carácter de un comando especial. Como la mayoría de las impresoras actuales van controladas por un microprocesador, es muy simple comprobar si cada carácter que llega es el carácter de escape - ASCII 27. Si lo es, el microprocesador de la impresora interpreta el siguiente o los siguientes caracteres como parámetros de la secuencia de escape. Estos parámetros hacen que se tomen ciertas acciones predefinidas. Las más típicas son:

> Imprimir subrayados todos los caracteres que reciba desde ahora.
>
> Imprimir todos los caracteres recibidos desde ahora con tinta más oscura (esto es en negrita).
>
> En las impresoras de matriz de puntos, puede haber secuencias de escape para que impriman todos los caracteres siguientes a doble altura de la normal, o a doble ancho.

Hay muchas otras facilidades en las impresoras. Obviamente, para cada facilidad especial debe haber una secuencia de escape que la active y otra que la desactive.

El único problema con las secuencias de escape es que no hay una forma estándar definida. (Lo más cercano es el ANSI Estándar). Esto significa que debe tener cuidado de comprobar el manual de la impresora que compre. Asegúrese que la impresora es capaz de proporcionar facilidades especiales de impresión, y que tiene la información necesaria para usarlas. Tenga en cuenta que hay algunas impresoras que no tienen facilidades especiales. Muchos paquetes de proceso de texto permiten construir sus propias secuencias de escape para adaptarlos a

una impresora en particular. El paquete AMSWORD, escrito especialmente para que se ejecute en su CPC, le permite definir hasta 40 secuencias de escape. Esto es más que suficiente para casi todas las impresoras más sofisticadas. La regla de oro, cuando se usan secuencias de escape, es leer el manual de la impresora, las secuencias de escape son muy diferentes de una a otra máquina.

Una palabra final sobre las secuencias de escape. En el mercado hay muchas VDUs de segunda mano, a buenos precios. Si compra una de estas, asegúrese de conseguir una copia del manual. La mayoría de los fabricantes de VDU después de 1977 incluyen alguna secuencia de escape. Si va a conectar una de estas VDU a los interfaces serie descritos en el capítulo tres de este libro, puede implementar los efectos de estas secuencias de escape para mejorar sus juegos o gráficos. Los VDU, como el DEC VT52 o el Systime 5753, tienen muchas de estas facilidades. No pueden igualar la
capacidad gráfica de su pantalla del CPC, pero pueden serle útiles en ciertas aplicaciones.

Ejemplo de una secuencia de escape

Para imprimir subrayado en una impresora Epson FX80, desde el BASIC, debe usar la secuencia:

```
PRINT #8,CHR$(27);CHR$(27);CHR$(45);CHR$(49)
```

Esta secuencia activa la facilidad de subrayado. Para desactivarla debe usar la secuencia:

```
PRINT #8,CHR$(27);CHR$(45)CHR$(48);CHR$(32)
```

Zócalo para la impresora

Patilla 1 $\overline{\text{STROBE}}$	Patilla 19 GND
Patilla 2 D0	Patilla 20 GND
Patilla 3 D1	Patilla 21 GND
Patilla 4 D2	Patilla 22 GND
Patilla 5 D3	Patilla 23 GND
Patilla 6 D4	Patilla 24 GND
Patilla 7 D5	Patilla 25 GND
Patilla 8 D6	Patilla 26 GND
Patilla 9 GND	Patilla 27 GND
Patilla 11 BUSY	Patilla 28 GND
Patilla 14 GND	Patilla 33 GND
Patilla 16 GND	Todas las demás: N/C

Fig AP7.1 Conexiones de la puerta de impresora

INDICE

Otros libros AMSTRAD publicados por RA-MA:

-AMSTRAD CPC-464 PROGRAMACION AVANZADA: Guía del Usuario.
M. Harrison.
Abril 1985 - rústica - 153 págs - Ptas. 1.400,-

-TECNICAS DE PROGRAMACION AVANZADA CON AMSTRAD.
K. Hook.
Octubre 1985 - rústica - 161 págs.- Ptas. 1.600,-

-AMSTRAD CPC-464/664 y 6128 PROGRAMACION ESTRUCTURADA.

S. Raven.
Diciembre 1985 - rústica - 174 págs.- Ptas. 1.700,-

-APRENDE LOGO CON AMSTRAD. Ficheros en Castellano.
Spen, S.A.
Mayo 1986 - libro de 104 páginas + disco traductor del Logo ingles.- Ptas. 2.500,-

-DOMINE EL CODIGO MAQUINA EN SU AMSTRAD CPC-6128/664/464.
C. Gifford, y S. Vincent.
Junio 1986 - rústica - 228 págs - Ptas. 2.075,-

-RUTINAS EN CODIGO MAQUINA PARA SU AMSTRAD.
C. Gifford, y S. Vincent.
Mayo 1986 - rústica - 92 págs - Ptas. 1.200,-

-EL DOMINIO DEL AMSTRAD PCW-8256/8512.
J.M. Hughes.
Julio 1986 - rústica - 280 págs - Ptas. 2.360,-

EN PRENSA:
==========

CP/M GUIA DEL PROGRAMADOR. CP/M Plus, 2.2 y 1.4
A. Clarke, J.M. Eaton, D. Powys-Lybbe.

-GRAFICOS AVANZADOS CON AMSTRAD.
R. Ransom
Este libro cubre desde el dibujo de figuras sencillas, hasta aplicaciones tan avanzadas como las representaciones de moléculas.

-OBTENGA EL MAXIMO RENDIMIENTO DE SU AMSTRAD/
W. Johnson.
El libro incluye una colección de 100 subrutinas Amstrad para resolver todos sus problemas de programación.

AMSTRAD CPC-HARDWARE

La novedad del uso de los microordenadores solamente para juegos o programas basados en la pantalla pasará muy pronto. Usted pensará lo bueno que sería usar el micro para hacer algo útil.

Pero las cosas más útiles necesitan usar el micro para "escuchar" el mundo exterior, y como resultado tomar acciones determinadas. Aquí es donde se deben añadir elementos extras de "hardware" al 464, 664 o 6128 y usted debe saber cómo controlarlo.

Así que, este libro comienza con un aprendizaje fácil pero profundo para comprender como funciona el ordenador Amstrad y el procesador Z80 que hay dentro de él. Aprenderá rápidamente dónde están situadas las piezas de "hardware" más importantes -la RAM, la ROM, el chip 8255, el controlador de vídeo (VDU) y otros chips especiales del Amstrad. Con estos conocimientos tendrá un control completo de todo el "hardware" interno y estará preparado para pasar a la siguiente parte del libro.

Aquí es donde comienza la diversión. Usted usará sus conocimientos del Amstrad más algunos componentes baratos en proyectos excitantes que incluyen:

- Un teclado externo
- Una interfase RS232 de 4 canales
- Una tarjeta de expansión de ROM
- Un sintetizador de voz

Los proyectos avanzados para los más arriesgados incluyen:

- Un programador completo de EPROM
- Una red de área local para su oficina, fábrica o escuela

¿Le asustan sus pocos conocimientos de electrónica? No los necesita -no sólo se explican los conceptos necesarios, sino que los lectores pueden obtener los circuitos impresos y los kits completos de HALSTEAD DESIGNS.

Sobre el autor

Alan Trevennor es un ingeniero de Servicio de System Computers PLC.

ISBN 84-86381-19-3

ra-ma EDITORIAL
CARRETERA DE CANILLAS 144 28043 MADRID

AMSTRAD CPC-HARDWARE

ALAN TREVENNOR

'ccl-rna

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/