AMSTRAD

Manual de instrucciones para la conexion de las expansiones de memoria RAM de 64K Y 256K

EDITION 1

PRECAUCION: **Asegurese de mantener desconectado su AMSTRAD antes de conectar el interface al bus de expansión. De lo contrario, se puede causar un dano permanente al paquete RAM o al ordenador.**

CONTENIDO

1 Conectando al paquete RAM

Desconecte su ordenador AMSTRAD. Conecte el paquete RAM al enchufe de la parte trasera del ordenador. En el CPC 464 esta entrada se llama "Floppy Disc", en el CPV 664 y CPC 6128 esta se llama "Expansión". En el bus de expansión de la parte trasera del paquete RAM se pueden conectar expansiones tales como el interface de disco para el CPV 464, el lapiz de luz y sintetizador de voz de Dk'tronics, o expansiones ROM.

Ahora conecte el ordenador. Este debe encenderse normalmente. Si esto no ocurre, compruebe que todas las conexiones esten hechas correctamente. Todos los productos Dk'tronics tienen una situación de tecla en el conector para impedir problemas de alineamiento. Otros interfaces pueden no tener este sistema de tecla (el caso mas conocido es el interface de disc de Amstrad). De ahí que la causa de este tipo de problemas esta entre el paquete RAM y estas expansiones. Si éste es el caso, intente conectar de nuevo los interfaces antes de insertar el paquete RAM al ordenador. Esto le dara más visión al alinear los pins.

Si el ordenador no enciende o no functiona correctamente (distintos patronas en la pantalla), es posible que el monitor puede haber cortado la corriente del ordenador.

En caso de que el monitor sea de color, desconectelo e intente la conexión de nuevo tal y como se indica arriba. El monitor monocromo deberá desconectarse unos segundos hasta que la corriente llegue al ordenador.

ES muy dificil qqu el ordenador falle al encenderse si solo esta conectado el paquete RAM. Si este es su caso, la falla probablemente sea del paquete RAM.

Devuelva el paquete RAM a Dk'tronics si es este su caso.

2 Usando el RAM adicional

Hay dos formas de usar el RAM adicional. En el cassette adjunto al paquete RAM se continen algunas extensiones al BASIC. En este caso, el RAM adicional puede ser usado simplemente para programas BASIC. Alternativamente, el RAM es accesible ya sea por BASIC y codigo máquina utilizando el comando OUT. El programador experimentado sabrá usar el RAM para aquello que necesite y escribir software adaptado a su propósito. Los programas comerciales usaran sin duda este acceso o via.

El segundo método se describe detallademente en el capítulo 10. El primer método del uso del paquete RAM se explica a continuación:

Una vez que el ordenador este listo para funcionar, cargue el cassette de software RSX adjunto al paquete RAM.

a) En sistemas de disco teclee "|TAPE" y pulse ENTER. (recuerde que el signo "|" esta en la tecla "@").
b) Teclee "RUN" y pulse ENTER.
c) La secuencia de carga esta descrita con detalle en el manual de usuario de su Amstrad.
d) Cuando el programa haya terminado de cargar, se le indicará que introduzca una dirección de memoria. Por ahora, simplemente pulse ENTER. (Ver capítulo 11).
e) El ordenador probará el RAM y le indicará el RAM disponible, y la memoria del ordenador estará libre para sus propios programas.

El cassette contiene el mismo programa por ambas caras, por lo que si una: cara falla al cargar, dispondrá de la otra.

El resto de los programas del cassette son extractos del manual que pueden ser cargados desde la cinta si no quiere teclearlos.

3 Prueba de RAM

Cuando se carga el código RSX, este hace una prueba de RAM. Si este no funciona correctamente, el programa detectará el error y le informará. Además, imprimirá la información de diagnóstico para ayudarle a reparar el paquete RAM.

En el caso poco probable de que se detecte un error, anote la información recibida por el ordenador y devuelva el paquete RAM para su reparación o recambio.

4 Comandos BASIC extendidos

Existen un total de 12 comandos en el cassette RSX. Unos tendran parámetros, otros no. Algunas veces el comando podra tener diferentes formatos y números de parámetros. Hemos intentado explicar cada comando en la forma mas simple y en los siguientes capítulos describiremos mas parámetros que hacen los comandos más flexibles y económicos para la memoria. De todas formas, los usuarios poco experimentados preferiran pasar de largo algunas secciones quizás innecesarias al leer el manual por primera vez. Estas secciones innecesarias estarán marcadas con un asterisco (★).

Los nuevos comandos van antepuestos de una barra vertical "|". Este caracter esta en la tecla del caracter "@", que se encuentra directamente a la derecha de la tecla "P".

Probablemente haya notado que durante la prueba RAM, el ordenador imprimió el número del "banco" que estaba probando. Cada banco tiene 16K de memoria. En el expansor de 64K hay 4 bancos. El paquete de 256K RAM tiene 16 bancos. Para tener acceso a alguna zona en particular de la memoria del expansor debe haber un número de banco y posiblemente una dirección de banco.

Por ejemplo, teclee:
"|SAVES, 1" y pulse ENTER

El ordenador responderá imprimiendo READY. Al hacer esto, se ha almacenado lo que estaba en la pantalla en el banco 1.

Ahora, borre el contenido de la memoria usando CLS. Para obtener el contenido de la pantalla de nuevo, teclee:

"|LOADS, 1" y pulse ENTER

Puede salver tantas pantallas como le permita la memoria. Esto es, cuatro pantallas en el RAM de 64K y dieciseis en el RAM de 256K.

Estas pantallas pueden ser creadas a partir de otro programa o ser dibujadas utilizando un "light pen". Almacene estas en cassette o disco y cárguelas de nuevo en el RAM para ser usadas en el programa. Aquellas pantallas que tardan en generarse en un programa, por ejemplo, laberintos, pueden ser creadas una vez y almacenadas para su uso inmediato cuando sea necesario.

Estos comandos pueden ser resumidos:

|SAVES, [banco]	Salvar datos del banco
|LOADS, [banco]	Cargar datos del banco

5 Ventanas y menus pulldown

Una de las caracteríisticas que hasen que las ventanas del Amstrad sean menos flexiblas que las de ordenadores de gestión más grandes, es el hecho de que el contenido de una ventana que se solapa a la otra se pierde al usarse la otra ventana.

Hay dos neuvos comandos que permiten que el contenido de una ventana sea almacenado y vuelto a cargar de la RAM. Esto permitirá el uso de verdaderos menus pulldown, que pueden cubrir el texto, pero ne quitarlo

EJEMPLO 1:

```
10 MODE 1
20 FOR I=0.05 TO 1 STEP 0.05 : REM DIBUJO POR PANTALLA
30      MOVE 640*I,0 : DRAW 640*I,400
40      MOVE 0,400*I : DRAW 640,400*I
50 NEXT I
60 WHILE INKEY$="" : WEND : REM ESPERA QUE SE PULSE UNA TECLA
70 WINDOW #1,INT( RND(1)*19+1),INT(RND(0)*19+INT
   (RND(1)*5+17)), INT (RND(1)*14+1), INT (RND(0)*14+INT
   (RND(1)*10+5))
80 PEN #1,2:PAPER #1,3
90 |SAVEW,1,1:REM SALVAR CONTENIDO DE VENTANA AL RAM
100 CLS #1:REM LIMPIAR VENTANA
110 WHILE INKEY$="" "":REM ESPERA QUE SE PULSE UNA TECLA
120 PRINT #1, "ESTOS ES UNA VENTANA"
130 WEND
140 |LOADW,1,1:REM RESTURAR CONTENIDO DE VENTANA
150 GOTO 60
```

El programa anterior utiliza dos nueves comandos: |LOADW y |SAVEW. Como probablemente ya sepa, hay oche ventanas (0 a 7) que pueden ser definidas. El primer parámetro as la referencia de una ventana y el segundo es el número de banco.

|SAVEW, [número de ventana], [banco] almacena la ventana al banco

|LOADW, [número de ventana], [banco] carga la ventana desde el banco

Vea los capítulos sobre ventanas más detalladamente en el manual del usuario.

5a **Máe sobre ventanas (★)**

Une ventana de cualquier tamano, incluso de toda la pantalla, puede caber en un sólo banco del RAM de expansión. Este esta bien si la ventana ocupa casi toda la totalidad de la pantella o varia de tamano como en el ejemplo anterior. Por otro lado, si la ventana fue definida como de 10 × 10 en MODE 1, la memoria necesaria para almacenar esta ventana sera menor de 16K. De hecho, sólo se necesitan 1.600 bytes, por lo tanto utilizar un banco completo supondría perder aproximadamente 14K de memoria.

Para resolver este problema, el comando de ventana RSX puede definir un parámetro adicional para definir el lugar del paquete RAM donde desee que resida el contenido de la ventana. En la ventana de 10 × 10 podrá colocar los datos en cualquier sitio entre 0 y 14783. El comando se escribe de la siguiente forma:

|SAVEW, [número de ventana], [banco], [dirección de banco]

|LOADW, [número de ventana], [banco], [dirección de banco]

La dirección de banco corresponde a una dirección entre 0 y 16383. La cantidad de memoria en bytes, utilizada para almacenar una ventana necesita ser tomado del valor superior y esto deja el límite entre la cual los datos pueden almacenarse. Si pone los datos en el fonso del paquete RAM, en la dirección 0, entonces la memoria desde la dirección 1600 hasta 16383 esta libre para otras ventanas o para otro conjunto de datos.

Como calcular el tamano de una ventana

Para tener más de una ventana por banco, necesitará saber cuanta memoria ocupará la ventana. Si la ventana varia en tamano entre dos límites, utilize el más alto de los dos. Dependiendo del modo que este usando, las figuras se calculan como se explica abajo.

En cado caso: X1 Es la coordenada X izquierda de mayor valor
X2 Es la coordenada X derecha de mayor valor
Y1 Es la coordenada Y superior
Y2 Es la coordenada Y inferior

MODE 0 TAMANO = (X2 − X1 + 1) ★ 4 ★ (Y2 − Y1 + 1) ★ 8

MODE 1 TAMANO = (X2 − X1 + 1) ★ 2 ★ (Y2 − Y1 + 1) ★ 8

MODE 2 TAMANO = (X2 − X1 + 1) ★ (Y2 − Y1 + 1) ★ 8

El ordenador dará un error si la ventana es demasiado grande para ocupar el espacio que usted le ha asignado. De la misma forma, si el tamano es mal calculado, las ventanas pueden solaparse en el banco causando efectos raros.

EJEMPLO 2:

```
 10 PEN 1:PAPER 0:MODE 1
 20 SIZE = 14 ★ 2 ★ 10 ★ 8
 30 LOCATE 1,13 : PRINT " 'n' PARA NUEVA VENTANA 'd' PARA QUITAR
VENTANA
 40 WINDOW 1,14,1,10:PAPER 3:CLS
 50 BANKADRESS = 0:level = 0
 60 PRINT #LEVEL, "WINDOW";LEVEL
 70 KEYPRESS$ = LOWER$(INKEY$)
 80 IF KEYPRESS$ = "n" THEN GOSUB 110
 90 IF KEYPRESS$ = "d" THEN GOSUB 190
100 GOTO 60
110 IF LEVEL = 7 THEN RETURN
120 LEVEL = LEVEL + 1
130 WINDOW #LEVEL,1 + LEVEL ★ 3,14 + LEVEL ★ 3,1 + LEVEL ★ 2,10 +
LEVEL ★ 2
140 |SAVEW,LEVEL,1,BANKADRESS
150 BANKADRESS = BANKADRESS − SIZE
160 PEN #LEVEL,0:PAPER #LEVEL,(LEVEL AND 1) + 1
170 CLS #LEVEL
180 RETURN
190 IF LEVEL = 0 THEN RETURN
200 BANKADRESS = BANKADRESS − SIZE
210 |LOADW,LEVEL,1,BANKADRESS
220 LEVEL = LEVEL − 1
230 RETURN
```

El programa anterior utiliza solo un banco de RAM pero todas las ventanas (8) estan definidas. La variable "nivel" se usa para determinar el nivel de las ventanas y la variable "dirección de banco" indica el próxima lugar libre en le banco de RAM.

Hay dos comandos generales de movimiento de datos para permitir que los datos del programa sean movidos del y al paquete RAM.

Estos dos comandos son:

[SAVED, [banco], [ubicación de comienzo], [longitud], [dirección de banco]

[LOADD, [banco], [ubicación de comienzo], [longitud], [dirección de banco]

El primer parámetro indica el banco que desea usar. La ubicación de comienzo es una dirección de memoria donde hay algunos datos. La cantidad de datos viene dada como longitud. Opcionalmente una dirección de banco puede venir dada de tal forma que permita el almacenaje de más de un tipo de datos en la RAM.

Es posible salvar todo tipo de datos utilizando estos comandos, pero primeramente explicaremos como salvar arrays numéricos simples, siendo estos los más faciles de entender.

Pongamos el caso de que su programa trata de control de stock de hasta 60 items. Puede tener un array de cadena que contenga el número de cada item que tiene stock.

Así, se usará aproximadamente 1K para los nombres y 300 bytes para las figuras de stock. ?Que pasaría si quiere actualizar el valor del stock cada semana y quiere guardar el último ano de stock en un registro? o incluso los últimos cinco anos. En este caso las figuras ocuparían aproximadamente 15K o incluso 75K.

Esto podría almacenarse comodamente en disco, o incluso en cassette para un ano de stock, y los datos estarían disponibles cada vez que hiciera falta hacer un cálculo, pero probablemente estará de acuerdo en que se gastaría mucho tiempo en leer los datos cada vez que se calculase una distribución para cada item.

Obviamente sería más facil cargar todos los registros en el RAM, y tener acceso a los datos inmediatamente:

En vez de definir un array de dimensiones "stock (60,52)" ocupando aproximadamente 15K de RAM que podría ser usado para programas, definia un array "stock(60)". Lea todos los datos desde disco semanelmente y almacene los datos desde disco semanalmente y almacene los datos de cada semana en el banco RAM. Para hacer esto, necesitará saber dos cosas. Una ? en que parte de la memoria se almacena un array?, y dos, ?cuantos bytes es necesario almacenar?.

1) ?Donde se almacena un array?

La dirección de cualquier variable puede encontrarse rapidamente usando "@" antes de una variable. Por ejemplo, dimensione el array anterior:

```
DIM stock (60)
```

Ahora teclee: `PRINT @stock(0)`

El ordenador contestará dando la dirección de la memoria donde se ha almacenado el primer elemento del array. Intente:

```
PRINT @stock(1)
```

El número que sale, será cinco veces más alto en valor. Esta es la dirección de la segunda variable.

El prefijo "@" funcionará situandose en frente de cualquier variable. El primer item de cualquier array es obviamente "@stock(0)". Si esta utilizando arrays multi-direccionales, el primer item será "@stock(0,0) o "@stock(0,0,0,0) dependien do del número de dimensiones.

2) ¿Que longitud tiene un array?

Primero, los distintos tipos de arrays ocupan distintos números de bytes por elemento. Para números reales, hay cinco bytes por elemento. Los arrays de números enteros ocupan dos bytes por elemento.

Los arrays de cadena son de longitud variable. Todo esto se explicará más tarde.

Por ejemplo, "stock(60) tiene un total de 61 elementos
"stock(60,52) tiene 61 ★ 53 elementos = 3233 elementos
"stock%(10,5,12)" tiene 11 ★ 6 ★ 13 elementos = 858 elementos

Para hallar la memoria total, multiplique el número total de elementos por la cantidad de memoria necesaria para cada elmento.

Por ejemplo, "stock(60)" ocupa 61 ★ 5 = 305 bytes
"stock(60,52)" ocupa 3233 ★ 5 = 16165 bytes
"stock%(10,5,12)" ocupa 858 ★ 2 = 1716 bytes

El array que estamos usando tiene 305 bytes de longitud, y empieza en @ stock(0).

En un solo banco del RAM podemos almacenar 305 bytes aproximadamente 53 veces. La dirección de banco comienza en 0 y aumenta en pasos de 305 bytes:

0 305 610 915 1220 1525 etc.

Deberemos almacenar la semana no. 1 en la dirección de banco 305, la semana no. 2 en la dirección 610 y así sucesivamente hasta 52 semanas.

Los datos usados con el propósito de hacer tests pueden escribirse en disco o cassette mediante el situiente programa. Una vez escrito el test de archivo, guardelo para utilizarlo mientras desarrolla su programa.

```
 10 OPENOUT "stock.dat"
 20 FOR WEEK = 1 TO 52
 30     FOR ITEM = 1 TO 60
 40         PRINT #9, INT(RND(1) ★ 3000 + 100)
 50     NEXT WEEK
 70 CLOSEOUT
 80 END
```

Ahora teclee "NEW" e intruduzca el siguiente programa:

```
  10 DIM stock(60)
  20 INPUT "leer archivo (s/n)";ANS$
  30 IF LOWER$(ANS$) = "s" OR LOWER$(ANS$) = "si" GOSUB 1000
  40 RESTO DEL PROGRAMA.
1000 REM subrutina para leer datos de disco.
1010 OPENIN"stock.dat"
1020 FOR WEEK = 1 TO 52
1030     FOR ITEM = 1 TO 60
1040         INPUT #9, STOCK(ITEM)
1050     NEXT ITEM
1060     |SAVED,4,@STOCK(0),61 ★ 5,WEEK ★ 305
1070 NEXT WEEK
1080 CLOSE IN
1090 RETURN
```

El almacenaje de cadenas podría venir por si solo si todas las palabras fuesen de la misma longitud porque así no habría desperdicio. Por ejemplo, un examen de palabras que usase palabras de cinco, seis y siete letras. Un banco de RAM puede usarse para cada longitud de palabra. Un programa cargador establecería los datos en el RAM, y otro podría encadenarse y usar hasta 36K de RAM para programar.

Un array numérico podría tambien almacenarse en el banco RAM para indexar las primeras letras y así contribuir a la velocidad de acceso a una palabra en particular.

El programa enterior puede usarse para leer el archivo de disco o cassette. Una vez que el archivo este en el banco RAM, el contenido se quedará ahí para usarse hasta que el ordenador se apague, o hasta que otros datos sean puestos en ese banco. Esto significa que los datos solo necesitan ser leidos una vez del disco, y entonces el programa puede ejecutarse de nuevo sin perder los datos. Esto puede ser tambien util si desea escribir un número de programas para utilizar los mismos datos.

Una vez que los datos estan en memoria, puede tener acceso a los datos de cada semana simplemente cargando de nuevo el array de stock. Anada la sección que damos a continuación para dibujar una gráfica de barras para una sección dada.

```
100 MODE 2
110 LOCATE 1,1
120 INPUT "cual item para analizar";ITEMNO
130 IF ITEMNO<1 OR ITEMNO>60 THEN 120
140 CLS:LOCATE 30,1
150 PRINT "grafica de barras para item";ITEMNO
160 LOCATE 10,25
170 PRINT "ene feb mar abr may jun jul ago sep oct nov dic":
    REM 3 espacios entre cada una
180 FOR LOOP=0 TO 4
190     LOCATE 1,24-LOOP*5
200     PRINT STR$(LOOP);"000"
210 NEXT LOOP
220 MASK 225:MOVE 60,368:DRAW 60,0:DRAW 61,0: DRAW 61,368 :
    MOVE640,24:DRAW 48,24
230 FOR LOOP=1 TO 4
240     MOVE 48,LOOP*80+24:DRAW 60,LOOP*80+24
250 NEXT LOOP
260 FOR WEEK=1 TO 52
270     IF WEEK/2=WEEK/2 THEN MASK 170 : ELSE MASK 255
280 |LOADD,4,@STOCK(0),61*5,WEEK*305
290 YCOORD=(STOCK(ITEMNO)/4000*320 ) AND 4092
300 FOR XCOORD=1 TO 11
310     MOVE 49+XCOORD+WEEK*11,YCOORD+26 :
        DRAW 49+XCOORD+WEEK*11,26
320 NEXT XCOORD
330 NEXT WEEK
340 GOTO 110
```


Si usted tiene un programa que utiliza toda la memoria del ordenador debido a la necesidad de usar un array grande, puede usar el banco RAM para almacenar datos sin necesidad de dimensionar un array.

Por ejemplo, si tiene un array bi-dimensional "sales%(366,30)" para almacenar las cantidades de ciertos tipos de stock que se venden cada día en un ano. Aunque este usando números enteros, el array usa aproximadamente 22K de memoria.

En vez de tener el array completo en la meoria BASIC, sa peude tener acceso a cada elemento una subrutina para leer un valor, y una para almacenar un valor.

```
100000 REM cargue 'store%'de la memoria del banco usando 'year' y 'type'.
10010 P=(YEAR*31+TYPE)*2
10020 BANK=1:IF P>=16000 THEN P=P-16000:BANK=2
10030 |LOADD,BANK,@STORE%,2,P
10040 RETURN
11000 REM copie 'store$' al banco usando 'year' y 'type'
11010 P=(YEAR*31+TYPE)*2
11020 BANK=1:IF P>=16000 THEN P=P-16000:BANK=2
11030 |SAVED,BANK,@store%,2,P
11040 RETURN
```

Se usan dos bancos, 1 y 2, y las variables "ano" "tipo" se usan para referenciar que elemento se requiere. En las lineas 10030 y 11030, solamente han sido movidos 2 bytes al banco RAM y desde al banco RAM, ya que usamos números enteros. El "*2" que hay en las lineas 10010 y 11010 reflejan el hecho de que un número entero se almacena en 2 bytes. Si se usasen variables reales, se necesitarían 5 bytes. Las lineas 10020 y 11020 deciden si el elemento está en el primero o segundo banco.

Si es necesario que el array se rellene con datos de disco o cinta, no hay necesidad de limpiar los valores a cero. Si quiere que todos los elementos esten a 0, la forma más facil es almacenar una pantalla en blanco en cada banco al comienzo del programa:

```
10 MODE 1:PAPER 0:CLS
20    SAVES,1
30    SAVES,2
```

6b

Almacenaje de cadenas

El mayor obstacule para almacenar cadenas as que pueden variar de longitud y pueden almacenarse en cualquier lugar de la memoria, incluso en un programa BASIC. Un método para almacenar arrays de cadena se explica abajo. De todas formas puede encontrar un método más facil para almacenar cadenas que el que vamos a explicar, cuando considere exactamente lo que quiere hacer.

Suponga que ha querido almacenar 500 nombres, de 20 cracteres cada uno. Se separa un banco en unidades de memoria de 21 bytes cade uno de forma que se pueda acceder aleatoriamente a las cadenes.

En cada segmento de 21 bytes hay una cadena. Esto significa que usaremos un total aproximadamente de 10K. Si usamos la variable "nombre para especificar la cadena que queremos entonces podamos introducir dos subrutinas, una para poner una cadena desde el banco 1 al "name $" y otro para almacenar el contenido de "name $" al banco RAM número 1:

```
20000 REM asignar 'name$' al numero de cadena 'name'
20010 B$="                    ":REM 21 spaces
  20020 |LOADD, 1, PEEK(@B$+1)+PEEK(@B+2)*256,21,NAME*21
20030 NAME$=MID$(B$,2,ASC(B$) ):RETURN
```

```
21000 REM almacene 'name$' en el banco como elemento 'name'
21010 B$=''                              '':REM 21 espacios
21020 MID$(B$,1,21)=CHR$(LEN(NAME$))+NAME$
21030 |SAVED,1,PEEK(@B$+1)+PEEK(@B$+2)*256,21,NAME*21
21040 RETURN
```

Una cadena imaginaria "b$" se usa para formar el elemento antes de almacenarse en el RAM. El primer carácter se establece a la longitud de "name$". Los siguientes 20 caracteres se encuentram en el lugar donde se ha almacenado el contenido de "name$". Entances se copian 21 caracteres en el banco RAM. Quando la cadena es recobrada, los caracteres son copiados y "name$" se ajusta a la longitud correcta mirando al primer carácter.

El almacenaje de cadenas podría venir por si solo si todas las palabras fuesen de la misma longitud porque así no habría desperdicio. Por ejemplo, un examen de palabras que usase palabras de cinco, seis y siete letras. Un banco de RAM puede usarse para cada longitud de palabra. Un programa cargador establecería los datos en el RAM, y otro podría encadenarse y usar hasta 36K de RAM para programar.

Un array numérico podría tambien almacenarse en el banco RAM para indexar las primeras letras y así contribuir a la velocidad de acceso a una palabra en particular.

7 Animación e imágenes (★)

Hemos visto como se pueden almacenar y volver a sacar pantallas y ventanas. La animación es el acto de poner imágenes en la pantalla tan rapidamente que los ojos vean algo en movimiento. Con 64K ó 256K de memoria se pueden almacenar pantallas completas, y luego puestas en la pantalla para crear animación.

Habra notado en la sección 4 que cuando una pantalla se carga a la pantalla, puede ver como aparece cada linea. Para ilustrarse, teclee el siguiente programa:

```
 10 MODE 1
 20 BORDER 0
 30 FOR COL=0 TO 3
 40     INK COL,0
 50 NEXT COL
 60 FOR COL=0 TO 3
 70     PAPER COL:CLS
 80 |SAVES,COL+1
 90 NEXT COL
100 INK 0,1:INK 1,6:INK 2,21:INK 3,13
110 PEN 1:PAPER 0
120 WHILE INKEY$=" "
130     FOR SCREEN=1 TO 4
140         |LOADS, SCREEN
150     NEXT SCREEN
160 WEND
170 END
```

El programa almacena cuatro pantallas con color al banco RAM, y luego las carga secuencialmente. Sin embargo, el efecto que se obtiene es un patrón con rayas.

Para crear una animación que se adapte al ojo humano, el ordenador necesita crear el despliegue de pantalla, y desplegarse al instante.

Las tres nuevas instrucciones que permiten esto son:

|LOW |HIGH y |SWAP

Antes de entender estos comandos, es necesario saber como usar la pantalla. La pantalla normal esta localizada en 49152. De todas formas el Amstrad es capaz de "var" una pantalla de cualquier sitio de la memoria en 16384 esta libre mientras que el margen de BASIC se ponga por debajo de 16384. Usando esto, hemos llamado a la pantalla original pantalla superior y la nueva pantalla, en 16384, se llama inferior. Para ir de una a otra, simplemente use:

|LOW para poner en acción la pantalla inferior
|HIGH para reajustar la pantalla superior
|SWAP para permutar de superior a inferiror y viceversa

Cada vez que se hace una permutación, se le indica al ordenador, y todos los gráficos y textos subsiguientes aparecen en la pantalla seleccionade.

Para usar esta facilidad de permutar de una pantalla a otra instantaneamente, los comandos de pantalla y ventana pueden tener un parámetro adicional que le indique al ordenador que cargue o salve los datos de y a la pantalla alternativa.

Esta nueva forma se puede escribir:

|SAVES, [banco] , [permutación]

|LOADS, [banco] , [permutación]

|SAVEW, [número de ventana] , [banco] , [dirección de banco] , [permutación]

|LOADW, [número de ventana] , [banco] , [dirección de banco] , [permutación]

Si es valor de permutación es cero, por omisión, entonces el comando actuará en la pantalla que se este mostrando en ese momento. Alternativamente, si el valor es uno, el ordenador cargará y grabare datos de la pantalle que no se esten mostrando. Cuando el trabajo ha sido hecho, el ordenador puede permutar pantallas, y el efecto es que la pantalla parace cambiar instantaneamente.

En al programa anterior teclee estas lineas:

```
  5 MEMORY 16383 :|HIGH
135 IF SCREEN/2=SCREEN/2 THEN T=TIME:WHILE TIME<T+20:WEND
140 |LOADS,SCREEN,1: |SWAP
```

Ahora que el ordenador puede construir la pantalla mientras se muestra otra, no hay patrón. La pantalla de color parece cambiar instantaneamente.

Dado que el banco de memoria se mueve en la dirección de espacio en 16K se invierte más tiempo para la transferencia de pantallas a la pantalla inferior que a la superior. Por lo tanto, la linea 135 demors al ordenador ya que debe cargar la pantalla superior. Esto significa que el tiempo que esta cada pantalla en pantalla se mantiene igual. Intente quitar la linea 135 para ver la diferencia.

Si se pone una demora mayor entre las lineas 140 y 150 se conseguiré el efecto de muestra de pantallas. Alternativemente se pueden seleccionar pantallas cuando se pulsa una tecla.

En pequena escala, se podría definir una ventana y los gráficos se podrían mastrar rapidamente sin recurrir a pantallas permutantes.

Hay que hacer notar, que es de poca utilidad el hecho de que el contenido de pantallas y ventenas pueden almacenarse de una pantalla que no se este mostrando

simplemente anadiendo un uno para el parámetro de permutación. Por ejemplo, si quiere cargar una serie de pantallas de cassette o disco, cárguelos en la pantalla inferiro (16K). Los mensajes generados por el sistema de cassette no necesitan ser desenchufados, ya que el contenido de la pantalla no se cambierá en la pantalla de baja meoria.

```
10 LOAD "pantalla 1", 16384 : |SAVES, 3, 1
```

Esta linea hara que se cargue una pantalla y se salve al banco 3. La pantalla que ve el usuario puede tener algo mas en ella.

8 Programación avanzada (★)

Esta sección introduce un nuevo comando y algunos otros aspectos de programación que pueden serle de utilidad.

El nuevo comando es:

|ASKRAM, [pregunta] , [variable]

El comando permite que ciertas constantes se encuentren por el programa que este escribiendo. Por ejemplo puede indicar el número de bancos libres pare el programma, ya que cambiará dependiendo de si el paquete es de 64K o 256K. El valor de la "pregunta" es un número de 1 a 3 que selecciona lo que usted quiere saber. La respuesta se coloca en una variable de números enteros definida por el segundo parámetro.

1000 a% = 0 : |ASKRAM, 1, @a% .. asignará a% a la cantidad de RAM.

1100 a% = 0 : |ASKRAM, 2, @a% .. asignará a% al número de bancos

1200 a% = 0 : |ASKRAM, 3, @a% .. asignará a% a 0 ó 1 dependiendo si hay un problema con al RAM.

El último comando puede usarse para aseguararse que el RAM está ahí y listo para usarse si en sus programas no quiere cargar el cargador RSX primero. Es posible cargar el código maquina por separado:

```
 20 MODE 1:PRINT "programa cargando"
 30 L=HIMEM
 40 MEMORY 9999
 50 LOAD "rsx", 10000
 60 L=L-(PEEK(10004)+ PEEK(10005)*256+1)
 70 POKE 10002,L-INT(L/256)*256
 80 POKE 10003, INT(L/256)
 90 PRINT CHR$(30);CHR$(21);
100 CALL 10000
110 PRINT CHR$(30);CHR$(6);
120 A%=0 : |ASKRAM,3,@A%
130 IF A% THEN PRINT "el RAM falla":END
140 CLEAR:MEMORY PEEK(10002)+PEEK(10003)*256-1
160 CHAIN "parte 2"
```

El programa anterior cargaré al codigo maquina RSX y lo pondré en memoria. No se imprimirá nada en la pantalla a menos que se compruebe que hay un fallo en al RAM o que no haya nada ahí. El programa "part 2" sería la mayoria del programa. Si se carga el programa en dos partes se puede ahorrar el tener que cargar el codigo RSX cada vez que el programa es ejecutado.

El codigo tiene que cargarse en 10000 en meoria antes de ser relocalizado para su uso. El valor de 16 bit en las localizaciones 10002 y 10003 corresponde al sitio donde quiere que el codigo se localize. Otro valor de 16 bits localizados en 10004 y 10005 contiene le longitud del codigo que se mueve en lo más alto de la memoria. Casi 1 K del programa se necesita solo una vez – la relocalización y los programas del test de RAM – y por tanto esta parte no se mueve a una zona mas alta del RAM.

Si dessea usar gráficos definidos por el usuario (UDG), anada las siguientes lineas:

```
 10 SYMBOL AFTER 256
150 SYMBOL AFTER 0
```

El valor en la linea 140 será diferente dependiendo de cuantos gráficos definidos por el usuario (UDG) desee.

Quizá quiera que en su programa haya un número diferente de estilos de grupos de caracteres. Despues de dar un comando SYMBOL AFTER, el HIMEM se situa justamente debajo de los gráficos definidos por el usuario. Por lo tanto es posible utilizar los comandos |LOADD y |SAVED para mover gráficos desde y a los caracteres gráficos.

Si tiene un programa que defina el grupo de caracteres, las definiciones pueden ser salvadas y cargadas al banco RAM para así tener múltiples grupos de caracteres.

```
 10 SYMBOL AFTER 0
 20 CHARS = HIMEM + 1
 30 REM definir simbolos aqui
```

Este programa salvará su grupo de caracteres a disco o cinta. En su programa final puede querer cargar un número determinado de grupos:

```
10 SYMBOL AFTER 0
20 CHARS = HIMEM + 1
30 LOAD "set1.grp",CHARS:|SAVED,1,CHARS,2048,0
40 LOAD "set2.grp",CHARS:|SAVED,1,CHARS,2048,2048
50 LOAD "Set4.grp",CHARS:|SAVED,1,CHARS,2048,4096
```

La razón por la cual la variable "chars" se establece es que el valor de HIMEM se altera cuando se accede mediante disco o cinta.

Durante el programa, se puede usar una subrutine para seleccionar un grupo de caracteres:

```
1000 REM dada la variable 'set', cargue los caracteres
1010 |LOADD,1,CHARS,2048,(SET-1)*2048
1020 RETURN
```

Observe que se usa la variable "grupa". En la secuencia de carga anterior grupos de 1 a 3 serán válidas. Se pueden anadir mas o menos de acuerdo a sus necesidades.

Todo este establecimiente puede hacerse en el programa cargador, solo una vez. Cuando 1 programa se ejecuta, no hay que volver a cargar el banco RAM.

El establecimiento de chars puede encontrarse en todo momento mediante:

200 CLEAR : SYMBOL AFTER 0 : chars = HIMEM + 1

Esto quitará cualquier buffer de disco que se haya establecido y "chars" sin duda apuntará a los caracteres.

9 Peekeando y Pokeando (★)

Hay dos comandos que permiten que la memoria de los bancos sea visualizada y cambiada byte a byte.

|POKE, [banco] , [dirección de banco] , [valor]

|PEEK, [banco] , [dirección de banco] , [variable]

|POKE trabaja de manera similar al POKE original. Necesita suministrar un número de banco además de la dirección normal y valor. La dirección de banco esta entre los margenes do 0 a 16383 o 0 a 16K.

|PEEK es un comando, a diferencia de la función normal. El banco y dirección de banco son las mismas que para |POKE. Para hallar el valor, necesita suministrar una variable de números enteros de la misma forma que en el comando|ASKRAM.

Por ejemplo:

```
10 VALOR% = 0
20 |PEEK, 3, 12345, @valor%
30 PRINT valor%
```

El programna anterior leerá el byte de situación 12345 en el banco número 3. El caracter " " indica a la extensión RSX en que parte de la memoria esta la variable para asi poder cambiar su contenido al byte requerido.

|PEEK y |POKE no son realmente comandos para un principiante. Es más, han sido incluidos para el programador más avanzado que desee usar el banco RAM de su manera.

Otro comando avanzado que se ha incluido para el programador avanzado es |BANK.

|BANK, número de banco

El comando viene seguido de un parámetro. Si este parámetro no esta presente, se asume que es 0. El banco al que se hace referencia está desde 16K a 32K. Un número de banco 0 situará el RAM original a su sitio. Desde el número 1 hasta el número mayor de banco situaran ese banco en el espacio de direcciones. Si un banco es situado en el mapa, el ordenador usará la memoria del banco en vez del RAM normal. De todas formas, la pantalla se seguirá tomando del RAM original si se ha usado |LOW. La ventaja de esto, es que la memoria completa puede usarse para programación en vez de tener que situar la parte superior de la memoria a 16383. La desventaja es que si el programa es detenido mientras se muestra la pantalla inferior, el ordenador escribirá datos de pantalla en el programa BASIC, causando un caos.

Asegurese conocer |BANK, |POKE y |PEEK antes de crear grandes programas utilizándolos. Salve el programa que este realizando cada cierto tiempo en caso de cometer un error y perder su trabajo.

Sin software RSX el programador podrá acceder a la memoria desde los bancos RAM. Para usar el RAM por usted mismo, se necesita cierto grado de conocimiento del mapa de memoria de su Amstrad.

Para ambos BASIC y codigo maquina, el bloque de memoria original desde 16384 hasta 32767 no puede usarse para programar. Por lo tanto, en BASIC necesitará establecer la parte superior de la memoria a 16383. En Código Máquina se puede utilizar toda la emoria excepto el bloque mencionado.

El RAM adicional esta situado en las direcciones 16384 hasta 32767 en 16 bancos. Una vez que el banco este situado, puede hacer con el RAM cualquier coas. No es indicado usar el banco RAM para código máquina porque si subsecuentemente se cambia el banco, !el programa desaparece!. En cualquier caso, los programas, pueden escribirse para ejecutarse en bancos y por lo tanto en el bloque original de 16K que se ha sacado, pero es necesario hacer el cambio de banco fuera de este margen de memoria. En BASIC seria extremadamente dificil usar el RAM del banco para un programa adicional, pero no imposible. Esa posibilidad se la dejamos a usted.

La manera en que son seleccionados los bancos se define abajo:

EN BASIC: Donde "banco" es el número de banco a situar.

OUT &7F00, 196 + (banco and 3) + (banco and 28) ★ 2

NOTA: Los números de banco son de 0 a 15.

Para re-establecer el banco original:

OUT &7F00, 192

EN CODIGO MAQUINA: Donde el número de banco esta en el acumulador. (A)

```
SELECCIONE: PUSH BC          ; seleccione banco A (salva todos los registros
                             excepto A 7 banderas)
LD C,A                       ;
AND 3                        ; (banco and 3) +
                             LD B,A
                             LD A,C
AND 28                       ; (banco and 28) ★ 2
                             ADD A,A
                             OR B
OR 196                       ; + 196
LD BC,07F00H                 ; BC = &7F00
                             OUT (C),A
                             POP BC
                             RET
```

De nuevo el número de banco en el acumulador comienza en 0. Para reestablecer el banco original:

```
RESTABLECER: PUSH BC         ; restablece la memoria original
             LD BC,07F00     ; BC = &7F00
             LD, A, 192
             OUT (C),A
             POP BC
             RET
```

La dirección de carga

El software que se carga desde cinta es relocalizable. De todas formas las areas de memoria en las cuales el programa puede ir estan limitadas entre 32768 y la parte superior de la memoria. Esto es asi porque el RAM banqueado aparece en el bloque 16384 hasta 32767. (Ver el capítulo anterior para averiguar porque). Por debajo del limite de 16K, la tabla de comando RSX no funcionará. Por lo tanto, durante la relocalización, el código se carga en memoria en 10000 y se mueve un espacio por encima en la memoria. Si se pulsa ENTER durante la carga, automaticamente se seleccionará la localización más alta disponible. Alternativamente, puede querer cargar el código en una dirección inferior y reservar algun espacio para sus propios programas.

Salvando el disco

El software del cassette no tiene protección. De ahí que para salvarlo a disco o incluso a otro cassette a la velocidad write 1, es cuestión de grabar los datos en la memoria, y despues salvarlo.

1) Teclee "|TAPE" y pulse ENTER (para sistema de disco).

2) LOAD "Bank"

3) MEMORY 9999

4) LOAD "rsx", 10000

5) Teclee "|DISC" o seleccionne SPEED WRITE como desee.

6) SAVE "bank"

7) SAVE "rsx", B, 10000, 4000

Compatibilidad con programas comerciales

El expansor RAM es compatible con el RAM banqueado que se suministra como estandar con el CPC 6128. Esto significa que una cantidad de programas escritos para el CPC 6128 funcionarán en el CPC 464 y el CPC 664.

De hecho, el software RSX suministrado funcionará en el CPC 6128 donde el expansor de 64K dará un total de 128K de RAM banqueado y el expansor de 256K dará un total de 320K de RAM banqueado.

Si algún programa falla al cargarlo en el CPC 464 o CPC 664, le sugerimos lo siguiente:

1) El software puede estar usando el vector de Firmware en &BD5B. Si este es el caso intente ejecutar el programa RSX antes de ejecutar el programa de aplicación.

2) El software puede chequear la ROM para determinar si se esta ejecutando en un CPC 6128. En este caso, cantactando con el fabricante del producto o con nosotros, puede alternarse el programa para que se pueda ejecutar en su ordenador.

3) El software puede usar algunas características del ROM CPC 6128, que no se encuentran en el CPC 464 y CPC 664. En este caso, contactando con el fabricante, podrá obtener información de como ejectuar el paquete.

Using CP/M

La operación del CP/M 2.2 es normal. No hay RAM adicional disponible para los programas de CP/M.

La utilización del CP/M 3 o CP/M + se explica en otro manual adjunto al expansor RAM.

Apendice **mensajes de error**

Al usar el software RSX, habré ocasiones en las cuales el ordenador no entiende, o no puede ejecutar aquello que se le ha ordenado. El software puede dar mensajes de error además de los mensajes normales. Los errores, y el porque se dan son explicados abajo:

1) Bad bank command (Mal comando de banco)	Se da si ha dado un número equivocado de pará metros o si no hay una variable donde debiese haber una.
2) Bank unavailable (Banco no disponible)	Ha intentado acceder a un banco que no está presente en su sistema.
3) Bad bank parameter (Mal parámetro de banco)	Ha referenciado un banco que nunca podrá caber en en ordenador.
4) Bad bank address (Mala dirección de banco)	La dirección que ha dado, está fuera de margen: los límites de direcciones de banco van de 0 a 16383.
5) Value invalid (Valor no valido)	La dirección de banco puede ser muy grande para el bloque de datos definido. El parámetro para |ASKRAM es distincto de 1, 2, o 3. El tamano de un bloque a salvar es mayor de 16K.
6) Bad window definition (Mala definición de ventana)	La ventana referenciada en |SAVEW o |LOADW está por encima de 7.

Apendice **Referencia de comandos RSX**

Todos los comandos adicionales se listan a continuación, como recordatorio de sus funciones y síntaxis.

PANTALLAS

|SAVES, [banco] , [permutación]
|LOADS, [banco] , [permutación]

VENTANAS

|SAVEW, [no. de ventana] , [banco] , [dirección de banco] , [permutación]
|LOADW, [no. de ventana] , [banco] , [dirección de banco] , [permutación]

BLOQUES DE DATOS

|SAVED, [banco] , [localización de comienzo] , [longitud] , [dirección de banco]
|LOADD, [banco] , [localización de comienzo] , [longitud] , [dirección de banco]

ANIMACION

|LOW	(Pantalla inferior).
|HIGH	(Pantalla superior).
|SWAP	(Alternación entre pantallas superior e inferior).

OTROS

|POKE, [banco] , [dirección de banco] , [valor]
|PEEK, [banco] , [dirección de banco] , [variable]
|BANK, [banco]
|ASKRAM, [pregunta] , [variable]
([pregunta], 1 = RAM, 2 = bancos, 3 = ?Ha occurrido error?)

DEFINICIONES

[banco]	no. de banco 1-4 o 1-6 para expansores de 64K y 256K.
[dirección de banco]	dirección dentro del banco 0 a 16383.
[permutación]	0 u omisión significa que actua en la pantalla presente, 1 significa que actua en una pantalla alternativa.
[dirección de comienzo] y [longitud]	Define un bloque de memoria original.
[variable]	Da la localización de una variable de número entero para ser asignado, por ejemplo @ b%.

PRECAUCION: **Asegurese de mantener desconectado su AMSTRAD antes de conectar el interface al bus de expansión. De lo contrario, se puede causar un dano permanente al paquete RAM o al ordenador.**

dkʼtronics Limited,
Longs Industrial Estate, Englands Lane, Gorleston, Great Yarmouth,
Norfolk NR31 6BE, England. Tel: (0493) 602926 (5 Lines).

Printed by Lowestoft Printing Co. Ltd., Walton Road Tel: (0502) 2502.