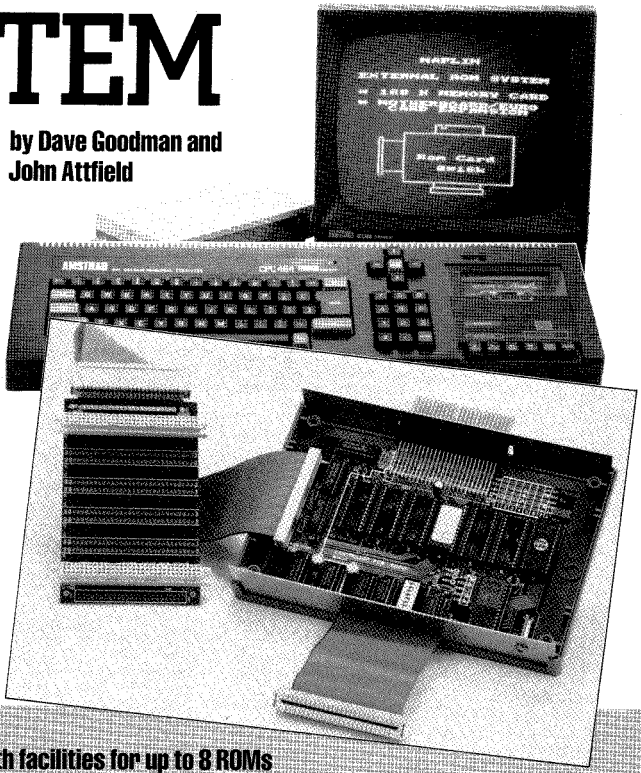
Price £1.50

XH65V

AMSTRAD EXPANSION SYSTEM

by Dave Goodman and John Attfield



Introduction

The Maplin ROM card allows you to add up to eight 2-16K ROMs on the Amstrad CPC 464, 664 and 6128. ROMs may be 2, 4, 8 or 16K in size and 'bytewyde' compatible EPROMs can also be used, providing they meet the Amstrad requirement of 200ns access time or faster. Many slower 250ns devices may well work but are likely to produce unpredictable results especially when PLOTing or DRAWing to the screen. So always try to use the faster versions for 100% reliability.

ROM Types

Maplin's range of 2716 to 27128 EPROMs can be used with the card as, depending on package size, they are pin compatible with each other. Most commercially available ROMs are of the 8K and 16K variety and pin configurated as the 2764 (8K) and 27128 (16K) EPROMs. It should be pointed out that some ROM/EPROMs are not directly pin compatible (Bytewyde) and will not run with this system.

Always ensure devices are correct before using them and meet the pin requirements in Figure 4 (more about this later).

- ★ **ROM Card with facilities for up to 8 ROMs**
- ★ **Accepts 2K(2716), 4K(2732), 8K(2764), and 16K(27128) EPROM types**
- ★ **Extension board and socket for DD1 Disk Drive (CPC464)**
- ★ **Buffering and mapped decoding for up to 128, 8-bit I/O addresses**
- ★ **Motherboard extension for plug in (Eurocard) modules**
- ★ **Mechanically and electrically compatible with CPC464, 664 and 6128 computers**

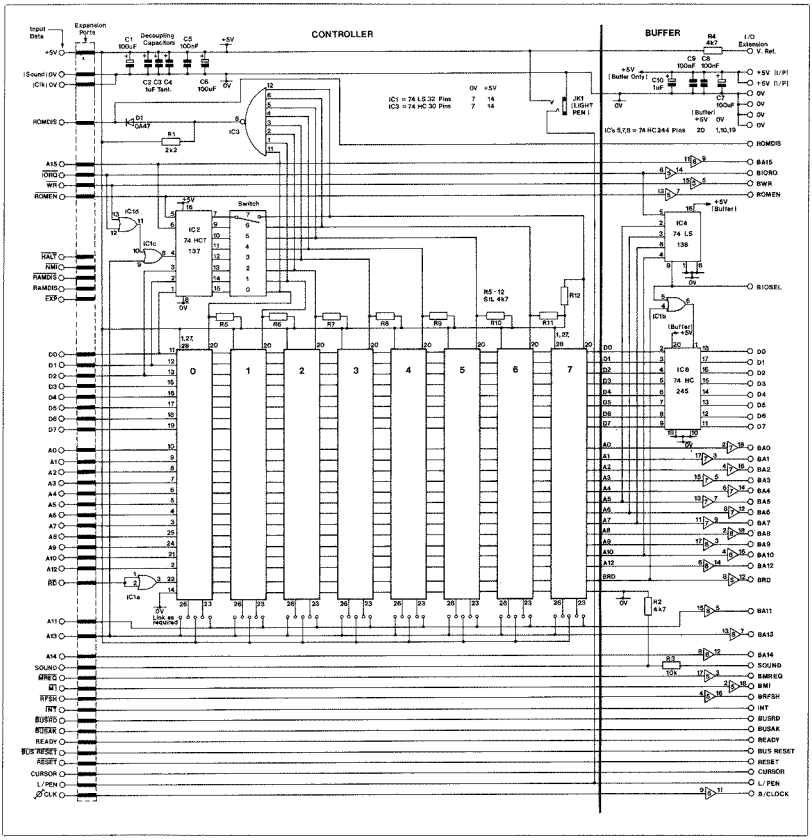


Figure 1. Circuit

Circuit Description

Referring to the circuit diagram of Figure 1, the layout can be viewed in two sections separated by a vertical, black line. The left section contains external ROM decoding components and the right section shows I/O buffering components. The external ROM section will be discussed first.

ROM Decoder

Facilities for extending ROMs have been provided in the Amstrad by the ROMEN (ROM enable) line, ROMDIS (ROM disable) line and Output Port address &DF00. All ROMs fitted (0 – 7 in this case) occupy the same memory space as the Amstrad video RAM (Random Access Memory) starting at address &C000. Obviously, video memory and extension ROMs cannot both be accessed at the same time, and it follows that none of the eight extension ROMs can be accessed at the same time, so a method of interrogation must be applied:

(A) To determine if and where any ROMs have been fitted.
(B) To perform this operation when the video display RAM is not being accessed.

ROM enable and disable lines help to realise this critical timing period under control of the Amstrad U.L.A. chip. Condition (A) is met during 'turn on' or system reset (EMS), where part of the initialisation routine addresses the I/O port at &DF00. At this port, data values 0 to 7 are written to IC2, which is a three to eight, latching decoder IC. IC1d and IC1c gate the IORQ, WR and A13 lines, and when ROMEN is active (low), a single data code or 'ROM position address' is presented to the one-of-eight decoder IC2. If an appropriate selection is made, then the associated ROM 0 – 7 (Figure 5) is selected, and the internal BASIC ROM is disabled via IC3 and D1.

ROM Classification

At this stage, it becomes necessary to digress a little into ROM classifications. Within the Amstrad the operating system and BASIC, although resident in a single ROM, can be viewed as two separate sections. The section containing BASIC can be switched in or out of use completely and usually resides at ROM address 0 (zero).

When BASIC is interrogated during EMS, the first data byte is examined and to be recognised as a BASIC ROM the byte must be &80. This Hex value tells the operating system that a Fore-

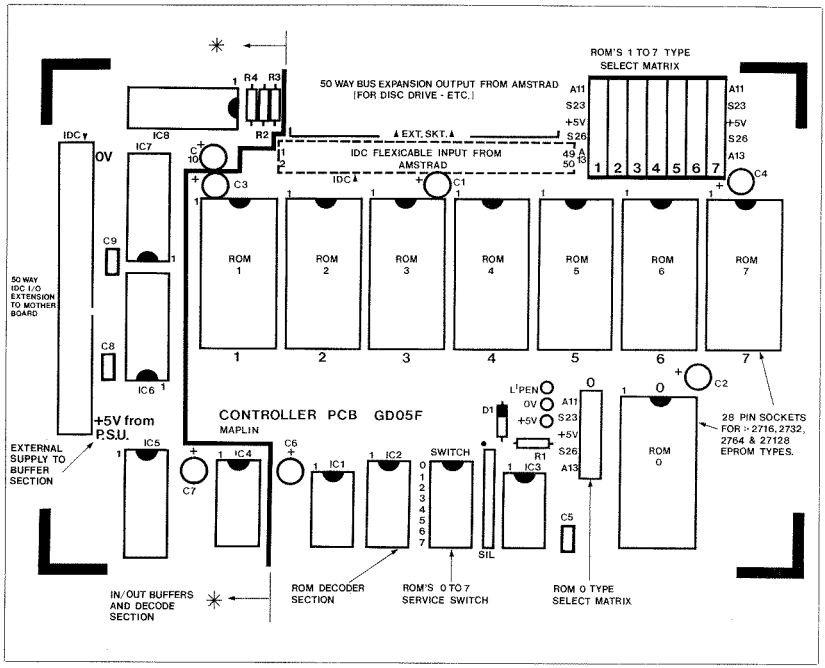


Figure 2. ROM Card overlay

ground program called BASIC is present, and provided that no other Foreground ROM is fitted, BASIC is entered and the appropriate message displayed on the monitor. A Foreground ROM contains the interpretive software and is always entered first.

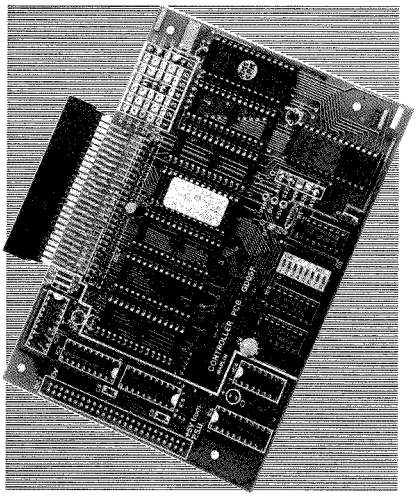
Unlike a Foreground BASIC program, other Foreground programs can receive priority and be entered first, thus replacing BASIC and taking over operation of the system. Examples of Foreground programs can be FORTH, PASCAL or Disk Utility ROMs, and many are commercially available. To receive entry priority over BASIC, the Foreground ROM must have its very first data byte value set at &00 (zero) and be fitted externally in ROM position 0. Further Foreground ROMs must be fitted from ROM position 1 upwards to position 7.

When a Foreground ROM is thus fitted at ROM address 0 (and S0 selected 'on'), it is entered and BASIC is transferred to ROM address 1, or, to the next unused address following further Foreground ROMs, if fitted.

Two other types of ROM can be fitted, known as Background ROMs and Extension ROMs. Both types function in a different way to Foreground ROMs, and a Background ROM has its first data byte set to &01, while Extension ROMs have the first byte value set to &02. These classification codes and ROM types are explained in greater detail further on in this booklet.

To continue with the circuit description, IC2 had previously decoded the data bus (D0 to D2) and an active low signal is presented via switch S0 to ROM 0 (if S0 is selected 'on'). At this time, the RD (read) line is active, and ROM 0 is read from address &C000. Classification codes are examined, and if none are available, a second data byte is written to Output Port &DF00 for decoding by IC2. In this way, all ROMs are interrogated - unless a Foreground ROM is found - and each one is 'LOGGED IN'.

A Foreground ROM fitted and switch selected at position 0 will 'Auto-Boot' in place of BASIC, and one of its first responsibilities is to



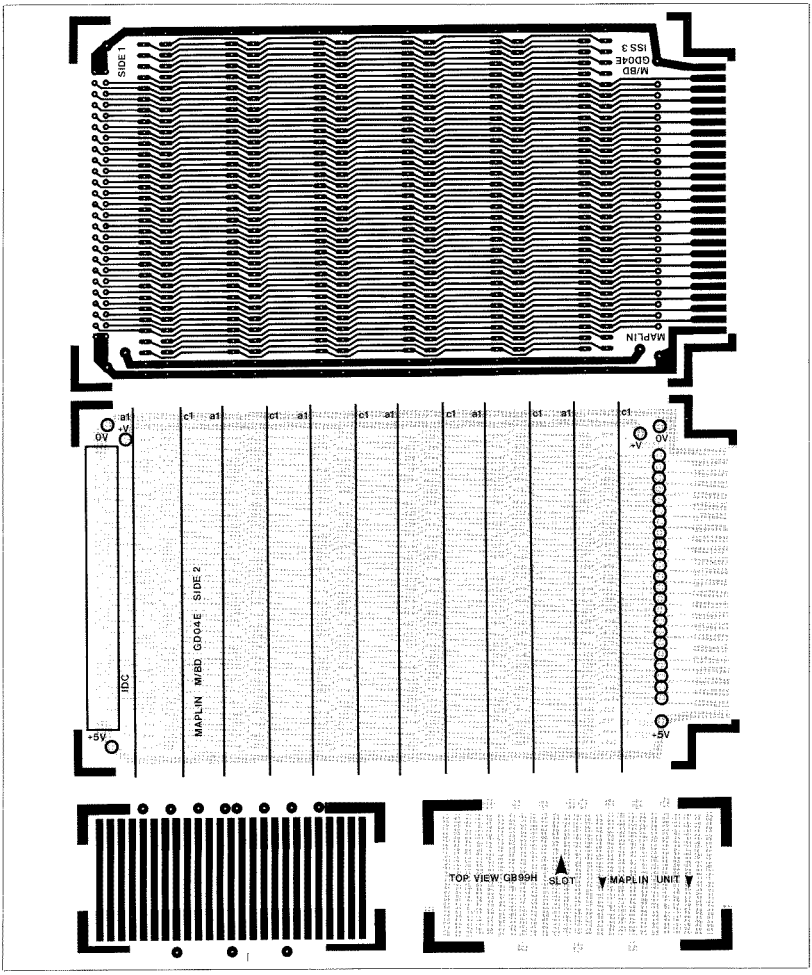


Figure 3. Motherboard and Extendiboard layouts

'LOG IN' any other ROMs that have been fitted. The eight way switch allows any of ROMs 0 – 7 to be selected during EMS only. If, for instance, eight background ROMs are fitted and switch S3 selected 'on' and the other seven switched off, then only ROM 3 will be logged in, the rest remaining inactive. Switch selecting another ROM, after turning on the Amstrad, will not automatically allow access to it unless a system reset is initiated.

ROM Type Selection

On the circuit diagram, each ROM position 0 – 7 has pin 23 and pin 26 shown unconnected. These pins can be associated with +5V and address lines A11 and A13, depending on the Type of ROM to be fitted. Figures 4 and 6 will help to show the requirements as follows:

(1) **2716-2K EPROM:** has twenty-four pins on the package and pin 1 should be inserted into pin S3 of the socket, so that pins 12 and 13 fit into the end socket positions S14 and S15 respectively. *Four unused socket pins should be left visible at the top.* This also applies to 2732-4K EPROMs.

(2) **2764-8K and 27128-16K EPROMs:** have twenty-eight pin packages and fit exactly into the socket positions. Socket pin numbers S23 and S26 are brought out onto solder pads along with +5V, A11 and A13 and must be linked appropriately.

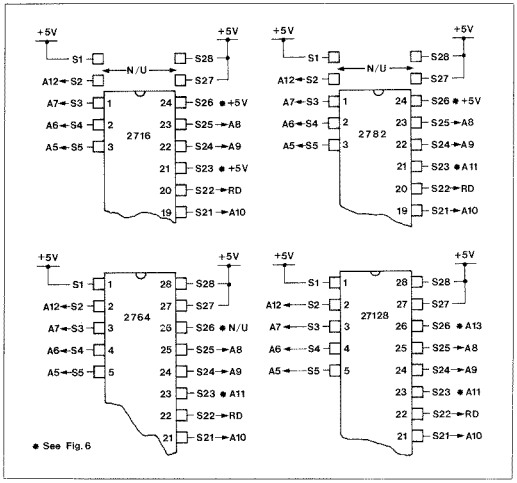


Figure 4. ROM types

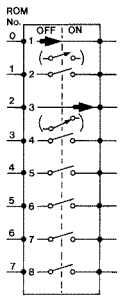


Figure 5. DIL switch

ROM Type Select Matrix

Figure 6 shows four different matrix connections for the four types of ROM which can be used on this card. Connect links between pads and solder them on the component side 2 as follows:

ROM Type	Matrix Links
2K – 2716	S23 to +5V and S26 to +5V
4K – 2732	S23 to A11 and S26 to +5V
8K – 2746	S23 to A11 only
T6K – 27128	S23 to A11 and S26 to A13

ROM 0 matrix is alongside the actual ROM socket position while ROMs 1 to 7 have their matrix at the top right hand edge of the board.

If you wish to fit ROMs having different identification markings, it is highly advisable to check with the supplier or a manufacturers data sheet to confirm that they are pin for pin compatible with the versions listed. *If they are not compatible then do not use them, as either ROM or computer may be damaged.*

Output Expansion

The ROM card has two expansion outputs, one comes directly from the Amstrad expansion port socket and can be used for 'add-ons' like the DD1 Disk Drive Interface (on CPC 464 only!), and the other output comes from the Buffer section.

Main CPU busses are buffered by IC's 4 to 8 for connection via an IDC transheader cable to the motherboard. The Amstrad has 128 I/O port addresses available for user purposes, specified as &F8E0 to &F8FF, &F9E0 to &F9FF, &FAE0 to &FAFF, and &FBE0 to &FBFF. No other port address should be used externally, and IC4 decodes these four block addresses from IORQ, A5 to A7, and A10 lines.

The data bus is buffered by a two way data selector IC6 with transfer direction (Read or Write) being controlled by the OR Gate IC1b. For example, writing a data byte &FF to I/O address &F8E0 would extend the Amstrad data bus (D0 – D7), via IC6, out to the motherboard, as BRD will be high from IC1b.

The byte &FF will be available on the data bus from IC6 outputs. The BIOSEL (Buffer I/O Select) control line is active low for any of the four block I/O addresses, and during a read cycle IC1b gate output is low reversing the data bus direction through IC6 to the Amstrad.

I/O Addressing

Amstrad have allowed a maximum of 128 user ports for external use as shown in Table 1.

The BIOSEL control line is active for any one of these addresses, but only during I/O request times. It remains inactive (high) during CPU memory cycles.

Light Pen

A 3.5mm stereo jack socket can be fitted and wired onto the ROM Card as in Figure 16. Nothing special has been added here, as the socket is only wired to +5V, 0V and the Amstrad L'PEN terminal on the Bus Expansion Socket, making access to this input easier. Light pens usually incorporate a photo-sensitive diode or transistor which, when placed onto the monitor screen, detects the raster phosphors as they are lit up by the CRT beam. Tiny pulses produced by the sensor are then amplified and shaped into definite square wave signals, which should be of a positive going waveform to be compatible with the L'PEN input on the Amstrad.

To use the pen facility requires specialised programming and an in depth knowledge of the 6845 display controller IC, as certain registers must be invoked for screen co-ordinate information. This information only relates to character cell resolution and not to pixel size, so it is somewhat limited for High-Res' graphics use.

Power Supplies

All components making up the controller section of the ROM card receive +5V and 0V supply directly from the Amstrad regulator. The buffer part of this project requires a separate supply which is developed at the motherboard end of the extension output, and is made available along the 50-way IDC cable. The PSU module (available May 1986, details in Vol. 5 Issue 19 of the 'Maplin Magazine'), produces various voltages to drive the buffer, together with future plug-in mod-

BINARY

A_{15} 1 1 1 1 1 0 X X 1 1 1 X X X X X A_0

A15 to A11 must be high.
A10 must be low.
A9 – 8 upper byte block address.
A7 to 5 must be high.
A4 to A0 lower byte address.

UPPER BYTE DECODING							LOWER BYTE DECODING						
(H)	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	(H)
F8	0	0	0	1	1	1	0	0	0	0	0	E0
F9	0	0	1	1	1	1	All 31 addresses					to
FA	0	1	0	1	1	1	available					
FB	0	1	1	1	1	1	1	1	1	1	1	FF

Available addresses are:
&F8E0 to FF, &F9E0 to FF, &FAE0 to FF, &FBE0 to FF.

Table 1

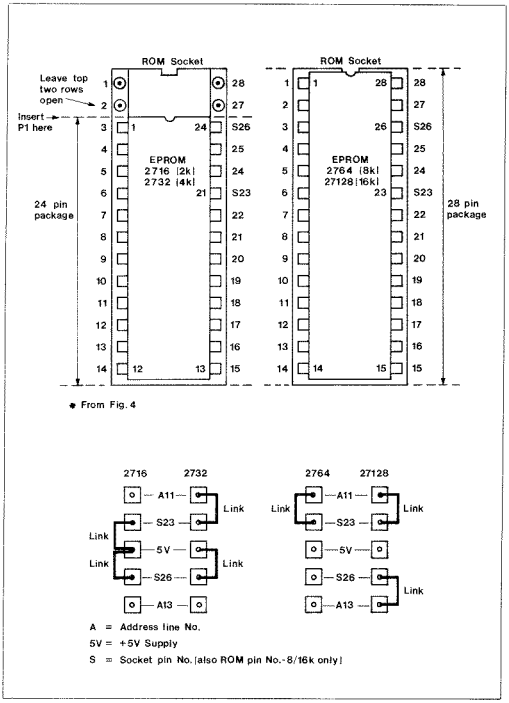


Figure 6. Matrix connections

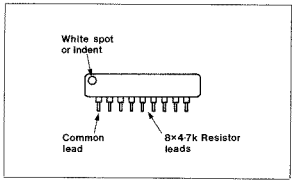


Figure 7. SIL resistor

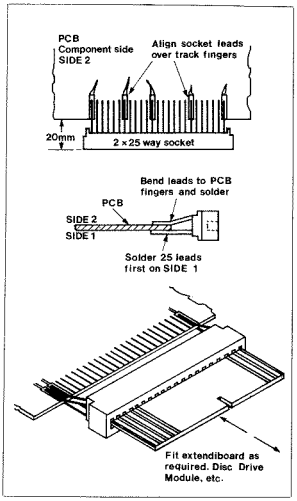


Figure 8. Expansion

ules fitted to the motherboard, but there is no reason why any +5V DC power supply cannot be used if so desired. Power for the light pen is also derived from the Amstrad, making a separate supply unneccessary for this device.

ROM Card Versions

Three versions of the project are available as follows:

(1) External ROM Controller Kit only.
(2) Motherboard Kit only.
(3) Ready-built Controller and Buffer.

Version (3) is available with all pcbs and cables assembled, but does not include a motherboard, track pins, veropins, or 64-way sockets and PSU.

The version (1) controller is available for adding up to 128K of ROM only. The buffering components and cables are not supplied with this kit, but the Bus Extension socket and Extendicard are included.

If you intend to expand the module for I/O use and will be adding further modules as they become available, then version (2) must be added. This kit contains all buffer components, 50-way IDC connector cable and motherboard. The 64-way sockets, plug-in modules and PSU are not supplied with the kit. (See current issue of the Maplin Magazine', or write for details of future projects.)

As an aid to testing the controller project, a pre-programmed EPROM is available (see Parts List), containing several RSX utilities. These routines are also described in the latter part of the section dealing with ROM programming, showing a complete external ROM assembly listing. The test ROM can be called from BASIC using the SHIFT @ symbol (¦).

Construction Details

ROM Controller Assembly

Locate and identify resistor R1, SIL resistor pack R5-R12 and diode D1. The SIL package is detailed in Figure 7. Fit R1 and the SIL array onto the pcb following the legend on side 2; a small spot at one end of the package must be orientated with the spot on the legend.

Fit D1 by carefully bending each lead both sides of the glass envelope. The glass body is easily cracked and caution should be exercised while doing this. Insert D1 with the black band lined up with the white bar on the legend.

Next mount the IC sockets in positions IC1 (14-pin), IC2 (16-pin) and IC3 (14-pin), with the slotted end aligned to the solid white block on the legend. This end denotes IC pin 1 position.

Bend a few leads over underneath the pcb on each socket, in order to prevent them falling out upon turning the pcb over for soldering.

Now mount eight 28-pin IC sockets in positions 0 to 7 and, likewise, orientate them to the legend and secure as before.

Now carefully solder all leads to the board, and snip off any excess wire ends. It may be advisable to solder one or two pins of the socket first to begin with, and then heat these with the iron whilst pressing the socket to the board to ensure that it is flat on the pcb.

Insert the octal SPST DIL switch, which can be any way around, but you may find it preferable to orientate it with switch 8 position towards the bottom edge of the board. Solder the switch in place.

Insert the miniature electrolytic capacitor C6 and PC electrolytic C1. The *negative leads* are denoted with a minus sign or bar, '–', on the package body, and the positive lead is usually the longest of the two.

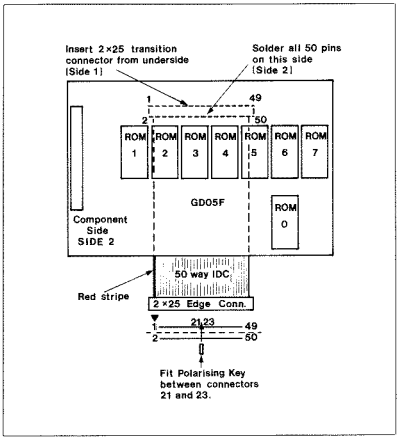
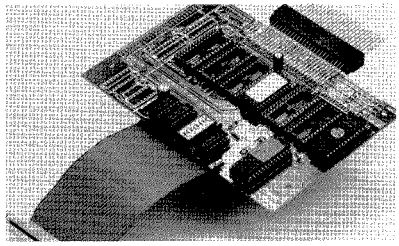


Figure 9. Fitting Transcable



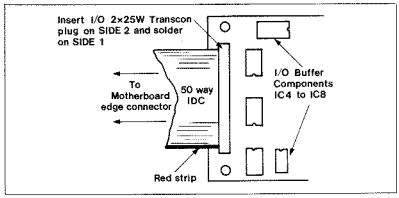


Figure 10. Connecting to motherboard

Fit tantalum capacitors C2 to C4, again taking note of the polarisation symbols (the positive lead is adjacent to the '+' sign), and finally, insert disc capacitor C5. Electrolytic or polarised capacitors *must* be fitted correctly according to their markings, as they will not function if reversed in circuit, and indeed may well violently burst open under these conditions!

Solder all remaining leads onto the board, and remove excess wire ends with side cutters.

Extending the Expansion Port

Refer to Figure 8 for pictorial details of this section. Take the 2 x 25-way edge connector socket, and slide the terminal solder pins (not the contacts) over each 'finger' in the area marked 'EXT. SKT' on the legend. On the soldering side (side 1) of the pcb, *carefully* solder both outside leads only, at this stage, to their respective positions as shown.

Now measure from the edge of the pcb to the *outermost* edge of the socket, and adjust the positioning by alternately re-heating the solder joints at each end until the *outside* edge of the socket is 20mm parallel to the pcb. Resolder both terminals until this measurement is approximately correct, then solder the remaining 23 terminals on side 1.

When completed, all terminals should lie flat onto the pcb, *side 1 only*. On side 2, use a screwdriver to bend each terminal down onto the 'fingers', so that they stay in this position without springing up when released. Again, solder all 25 terminals as before on side 2.

Before continuing, carefully check for shorts between terminals, using a resistance meter between all 50 leads to make sure this section is clear of any soldering errors. Also check for continuity between each track and its respective socket contact. Resolder dry joints and clear any solder bridges.

It is recommended that every soldered joint be cleaned with a suitable pcb cleaning solvent and stiff brush to remove solder splashes and flux. Doing so helps with inspection and fault identification which should be done at this point.

Short IDC Transcable

Figure 9 shows the slightly unorthodox method of fitting the short IDC cable for connecting the ROM Card to the Amstrad. The thin, 50-way header plug is fitted to the dotted IDC socket position from beneath the pcb on side 1 (on the soldering side), and soldered to the board on the component side (side 2). The red striped edge of the cable should be to the left and in line with pin positions 1 and 2 on the legend as shown. Both drawings should assist with this operation.

After soldering, clean the joints as before, and check for short circuits and continuity as previously described. The assembly is now ready to be

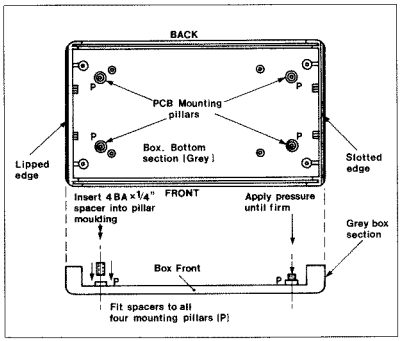


Figure 11. Mounting pillar positions

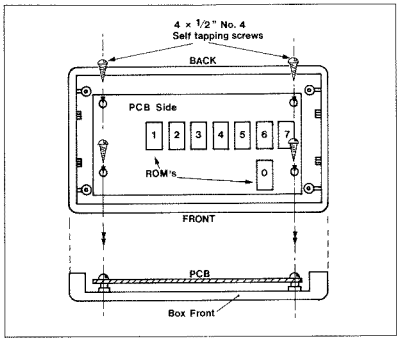


Figure 12. Mounting the pcb

installed into the case supplied, or to have the buffering components fitted.

Buffer and Motherboard Assembly

Locate and insert the three resistors R2 to R4. Mount four 20-pin IC sockets in positions IC5 to IC8, and a 16-pin IC socket in position IC4. Ensure the notched end of each socket lines up with each white block on the legend, and solder all component leads on the track side of the pcb, side 1.

Mount capacitors C7 and C10, again noting the polarising symbols for correct positioning as mentioned earlier, and fit both disk capacitors C8 and C9. Solder all remaining leads in position, and cut off excess wire ends.

Finally, fit the 50-way IDC transition header into the IDC Extension I/O position on the lefthand side of the pcb, but this time insert from the top or component side, side 2, of the pcb. The red striped edge of the cable *must* go to the end of the connector whose position is marked '+5V from PSU', and *not* to the '0V' end! If all is correct then solder all terminals on side 1.

Both sections of the ROM Card are now complete and ready for installation into the case.

Motherboard Construction

Not very much assembly work is required with the motherboard, as can be seen from Figure 14. There are 26 track pins and 3 vero-pins required for insertion into the pcb as shown. Push each track pin into the ringed holes on side 2, closest to the card edge connector. Each of 23 track pads and +5V, +V and 0V pads require a through-pin. Snap off each pin from the strip *after* insertion, and then use a hot soldering iron to push each pin head down onto the board. You must solder each pin to both side 2 and side 1 pads.

The three veropins are inserted into the holes 0V, +V and +5V at the IDC position end of the pcb. Fit the longer ends from side 1 and gently push them home. Solder with a hot soldering iron on *both* sides 1 and 2.

The motherboard can take up to 6 x 64-way receptacle sockets which are inserted from side 2. Note that stamped into the plastic body of each receptacle on the terminals side are the letters 'a', 'b' and 'c'. Insert these sockets with the 'a' and 'c' in alignment with 'a1' and 'c1' on the legend, as shown in Figure 14. If these sockets are fitted in reverse then

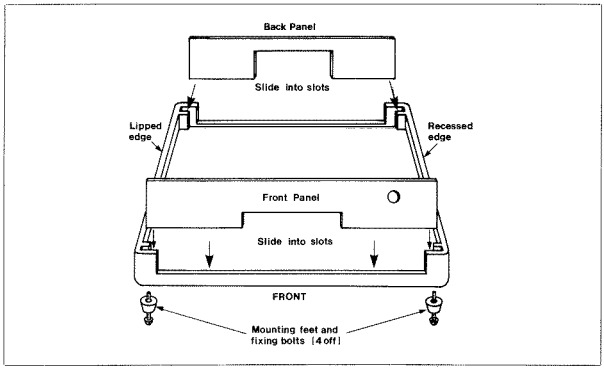


Figure 13. Fitting the Front and Rear Panels

modules inserted into them will be connected back to front – with disastrous results!

A 50-way IDC transition connection allows motherboards to be connected together using the short transheader cable as fitted to the controller pcb. When joining motherboards in this way, mount the transheader plug on side 2 with the red stripe of the IDC cable facing towards the bottom '5V' terminal pin. The 2 x 25-way plug will then fit onto the second motherboard card edge connector. Alternatively, connection can be made between boards using IDC or ribbon cable, by soldering each wire to both card edge finger and transheader hole position.

Note that the pcb side 2 fingers are common to the left-most row of 25 holes on all sockets on the pcb. The side 1 fingers are therefore commoned to the right-most rows of holes on all seven socket positions (as viewed from side 2). Power supply connections are made to the three veropins, and both 0V and +5V are extended along the ribbon cable to the buffer section of the ROM Card, and to the motherboard transheader position.

The +V terminal is not extended in this way, but is common to all 6 receptacle socket positions. Some row commoning between each receptacle position only exists on 'a3', 'c3', 'a4' and 'c4'. Also, 'a28' to 'a31' and 'c28' to 'c31' are commoned through to 6 positions only, but do not appear at any edge connectors. A Buffer-extension cable should be fitted via the 2 x 25-way IDC connector, with the red stripe along the cable at the bottom edge next to '+5V' track pin. A locating peg can be inserted into the IDC connector socket between the positions of pins 21 and 23 of the top row of pin numbers, and pins 22 and 24

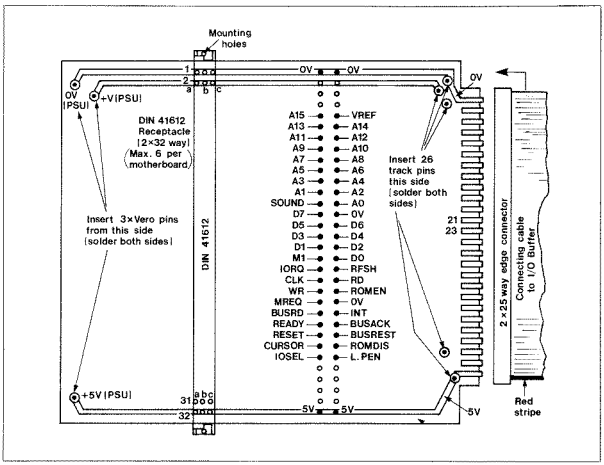


Figure 14. Motherboard pin functions

of the bottom row of pin numbers; which will line up with a slot cut into the motherboard card edge connector.

Figure 15 shows both ROM Card and motherboard terminal designations for reference purposes, and also shows veropin and receptacle fitting.

Box Mounting Assembly

The Vero box supplied with the kit is separated into two halves by removing 4 screws located in the feet. Refer to Figure 11 and position the bottom grey section with the slotted (recessed) edge to the right, and the lipped edge to the left. Press a ¼in. spacer into each of the 4 mounting pillar recesses – these are the inner pillars – and position the ROM Card as shown in Figure 12, with side 2 facing upwards, and the short IDC cable leading out towards the front of the box.

The Buffer IDC cable can be twisted underneath the pcb and brought out from the front or back of the box as desired, or it can be extended out from the left. It may be found necessary to run a file over the left hand, lipped edge of the box bottom section in order to avoid pinching the cable when assembling the box in this manner.

Secure the ROM Card by inserting 4 x self tapping screws into the pcb mounting holes, and through each spacer/pillar. Do not overtighten these screws as the pillars are only soft plastic. Slot a front panel over the IDC cable and fit the 3.5mm stereo socket (JK1) as shown in Figure 13.

The back panel can now be fitted. Wire the 3.5mm socket (JK1) to the light pen pads on the pcb, as shown in Figure 16, soldering the leads directly to the pads for L'PEN, 0V and +5V respectiveiy.

Testing

Do not fit any ICs at this stage. Ensure switches S0 – S7 are set in the 'off' position and, with the power turned off, insert the IDC 2 x 25-way expansion socket into the main expansion port at the rear of the Amstrad.

Switch on the computer via the monitor and check for a normal screen display message, and that key, tape or disk functions are working correctly.

Switch off power and insert IC1 (74LS32), IC2 (74HCT137) and IC3 (74HC30). Switch on again and check that all is as before.

If you have a suitably programmed ROM then fit this in

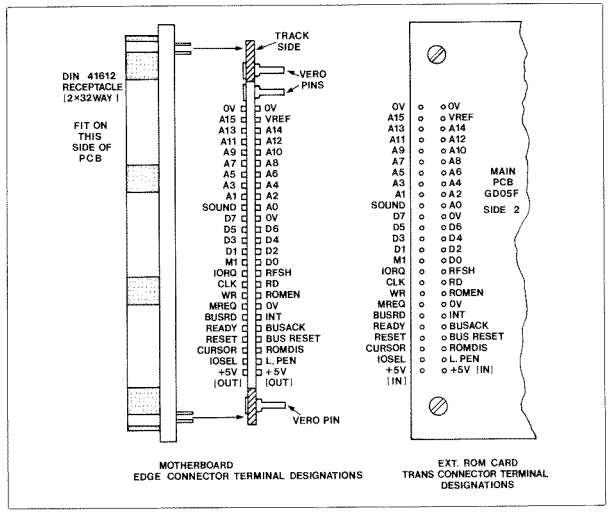


Figure 15. Motherboard and ROM Card terminal designations

accordance with the guidelines given in the paragraphs entitled 'ROM Type' and 'ROM Type Select Matrix' to be found near the beginning of this pamphlet (Figures 4 and 6), but *do not* fit any ROMs without first turning the power off!

If a CPC664 or C128 micro is being used, then *do not use ROM position 7 at all.* The same rule applies to CPC464 owners who are using a DD1 disk drive and interface module. 464 machines without a disk drive fitted can quite happily use ROM position 7 without any problems.

After determining the ROM size (2, 4, 8 or 16K), strap the matrix pads as necessary. If our Test ROM is to be used, then fit this into position 1, leaving socket pins S1, S2 and S27, S28 open. Strap the matrix 1 pads for a 2K-2716 type ROM (S23 and S26 to +5V) and select switch S1 (on the legend – not the body!) to the 'on' position. Turn on the power and a second message should appear on the display, after the Amstrad copyright paragraph:

"Maplin Test ROM"
BASIC 1.1

(Version numbers may differ).

Press the SHIFT and @ keys and type ¦ HELP followed by [ENTER].

The monitor should immediately produce the Maplin Test ROM Menu display as shown in Table 2.

```
M·A·P·L·I·N
XXXXXXXXXXX

AMSTRAD SERIES                    TEST ROM

ROM peek ..... ¦ ROMP, ADDR%, @VAR%
Reset inks ..... ¦ RINK
Scroll up routine ..... ¦ SCROLU, VAR%
Scroll down ..... ¦ SCROLD, VAR%
Wait ..... ¦ WAIT, VAR%
Epson screen dump ..... ¦ EPDUMP
Copy character ..... ¦ COPYC, X%, Y%, @VAR%
Screen base ..... ¦ BASE, VAR%
Screen offset ..... ¦ OFFSET, VAR%
Joystick 1 ..... ¦ JOY1, @X%, @Y%, @T%
Joystick 2 ..... ¦ JOY2, @X%, @Y%, @T%
This help screen ..... ¦ HELP

Press ENTER to return to BASIC >
```

Table 2.

The Maplin Test ROM message will always be displayed after EMS, provided it has been fitted and the appropriate positional switch selected. Try placing switch S1 (for ROM 1) in the 'off' position and reset the computer. The start up message will not be there! Type ¦ HELP as before, and the prompt "Unknown command" will be displayed.

As may be gathered from this, ROMs can be fitted, but made unavailable to the operating system if they are not switch selected, at any time.

If the Test (background) ROM is now fitted in position 0, or switch S0 is selected 'on' in addition to switch S1, then only part of the initialising message will be displayed after EMS (reset or turn on). This is because BASIC is being switched off, but at the same time is not being replaced by a Foreground program. Our Test ROM is programmed as a Background ROM!

Return the ROM and selector switches to normal and enter the BASIC extension command ¦ HELP for the menu again.

Twelve routines are displayed and the parameters required for accessing each routine are printed alongside.

Press [ENTER] to clear the display, and type ¦ SCROLD, 23. Both rows of print will immediately scroll down to the bottom of the display, paper area, after [ENTER] is pressed. Now enter ¦ SCROLU, 23. Both rows will scroll upwards to the top of the paper area. These extension commands can be used directly from the keyboard or from within BASIC and Machine Code programs. They occupy very little RAM space in the Amstrad, and BASIC programs can be typed in as normal with the ROM fitted and selected.

Some of the commands, as with SCROL, require information to be passed to the routine and others pass information back, as do the JOYSTICK routines. The ¦ RINK or 'Reset inks' command does not pass information back and forth, only acting directly on the Amstrad colour registers.

Return to BASIC if not there already, and change screen inks by entering INK 1,10 and INK 0,6. Obviously, green screen monitors will only show a change in luminance level, but colour screen monitors will show the effects well. With paper and pen inks altered, enter ¦ RINK. All colour registers will immediately return to their default values as they would normally after EMS.

Points to remember: Always link the matrix to suit the ROM being used (256K and 512K ROMs cannot be used). Use the appropriate switch to enable access to the ROM. Position zero should only be used for Foreground programs. Position 7 is normally used by the disk drive C.P.M. ROM. An explanation of Test ROM routines follows this section under the title 'Using Test ROM Routines From BASIC'.

If the Test ROM (or your own ROM) has been successfully accessed and used, then we can safely assume that the ROM Card is generally free of faults.

Amstrad CPC464 users who have a disk drive (DD1) may insert the extendi-card into the expansion socket at the rear of the ROM card, and plug in the disk interface module. Check that all disk functions perform normally. It may become necessary to clean the plug-in contact 'fingers' on the extendicard from time to time. A suitable pcb cleaning solvent or cellulose thinners does the job well when applied to the card with a soft cloth.

Continue testing that the ROM functions in positions 2 to 6 (and 7 if disk drive is not fitted), but remember to strap the matrix accordingly, and select only one switch (S0 – 7) 'on' at any time. Unfortunately, if you do not have a ROM to try in the card, then not much can be done to prove the circuitry is working, unless an oscilloscope is available.

To test the card with the aid of an oscilloscope, connect the ROM Card to the Amstrad after inserting the three ICs as described earlier in paragraph 3 of 'Testing'. Place all eight switches (S0 – 7) in the 'off' position, and type in the following BASIC program – after power-up of course!

```
10  J = 0
20  OUT (&DF00), J
30  GOTO 10
```

Place the scope probe on IC2 pin 15 (decoded ROM position 0) and RUN the program. A negative pulse signal of approximately 1 μS should be evident. Break the program (ESCAPE) upon which a +5V DC level is displayed. This test can be repeated for J values of 0 to 6 (not 7 if disk drive fitted!) in the program, and the IC2 decoder output (ROM position) checked for each value on pins 15 to 9.

IC1a pin 3 can be checked for a negative pulse $\overline{RD}$ signal from the CPU and also on ROM sockets 0 to 7, pin 22.

Testing the Buffer Circuitry

Fit ICs 4 to 8 into their respective sockets, with pin 1 orientated according to the legend, and insert the ROM Card into the Amstrad Expansion Port. Assuming that a previously assembled motherboard pcb is available, plug the long 2 x 25-way extension cable onto the motherboard card edge connector (Figure 14). The buffer section requires +5V and 0V to supply the ICs. Connect a suitable 5V DC source to veropins 0V and +5V (*but not +V*) on the motherboard.

Turn on the Amstrad power only, and check that everything functions as normal. Connect the negative probe of a voltmeter to the 0V pin 10 of IC8, and the positive probe to IC8 pin 20. A reading of +3V to +4V approximately should be obtained. Switch on the 5V buffer supply, and measure the voltage at IC8 again, which should now be the PSU voltage.

An oscilloscope can be used to view waveforms on all of the control and data lines (Figure 15). Some of the terminals, e.g. SOUND, RESET and BUS RD are not normally active, that is to say they are at constant voltage level (TTL), which does not necessarily mean that a particular buffer is faulty. Alternatively, the BIOSEL and data bus D0 – D7 can be checked with the aid of a diode (type 1N4148) and 8 x 10K resistors.

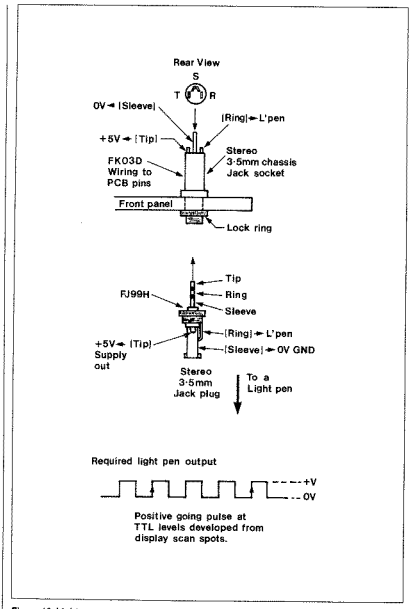


Figure 16. Light pen connection

For this method, switch off all power and wire one end of each resistor to each data line D0 – D7 in turn. This is best achieved by fitting the resistor vertically into the appropriate hole, from Side 2 of the motherboard at the IDC output position. D7 is the eleventh hole down along the left hand row, and D6 is twelve holes down on the right hand row (see Figure 15) from the positions of the 0V and +V pins.

Once all eight resistors have been soldered to the eight data lines, twist the remaining eight resistor leads together and terminate them at the PSU +5V terminal. Switch on all power again and type in the test program.

Test Program

```
10  MODE 1
20  LOCATE 18,12
30  PRINT 255 – INP
(&F8E0); CHR$(32)
40  GOTO 20
```

Run the program, whereupon '0' should be printed on the display. On side 1 of the motherboard, connect the cathode end (bar end) of the test diode to the BIOSEL track pin (first track pin up from the position of +5V track pin), and the diode's anode lead to D0. The number '1' should now be displayed.

Repeat this on D1 to D7, and check for D1 = 1, D2 = 2, D3 = 4, D4 = 8, D5 = 16, D6 = 32 and D7 = 64. Change program line 30 to input address &F9E0 and repeat the test. Input addresses &FAE0 and &FBE0 should also be inserted into line 30 to check that the IC6 decoder functions are correct.

Future Expansion

Euro card size modules will become available through 1986, details of which can be found in 'Electronics, the Maplin Magazine', or by writing to Maplin. Projects such as the PSU, the 6 x 8 (48 bit) I/O Card, and Eprom Programmer (for developing your own external ROMs!) will be made available in kit form and many existing projects such as the 'Video Digitizer' and 'Satellite Receiver' will be discussed for connection to the Amstrad computer using the ROM Card facility.

Using the Test ROM from BASIC

TERMINOLOGY

ADDR% ..... Integer Address.
VAR% ..... Integer Variable.
@VAR% ..... The Integer Address of a Variable.
¦ ..... The 'Shifted @' Symbol.
& ..... Represents a HEX Value.
% ..... Represents an Integer Value
# ..... Represents a HEX Value in Assembly Listings.

The integer variables X% and Y% have been used to represent screen coordinates, although they could just as easily have been any integer variable.

Introduction

A number of routines have been programmed into the 'Test ROM' that will enable the constructor to test the functions of the Maplin 'ROM Card' and illustrate the methods of passing information back and forth between the Foreground program (BASIC) and the ROM.

Each routine is called using the ' ¦ ' symbol (shift key and '@') followed by the command word, and where parameters are called for, these are tagged on to the end separated by commas.

Routines and Demo Programs

¦ HELP

Display the ROM menu on the screen.
For example, Just type in –

¦ HELP then [RETURN]

¦ ROMP, ADDR%, @VAR

Read a byte from this ROM at the address given. Use this routine to explore the ROM. See example 1.

¦ RINK

Reset all the inks to their default values. See example 2.

¦ SCROLU, VAR%

Scroll the whole screen up the number of lines to the value of the integer variable. The bottom line is cleared to the current paper ink. The top line will be lost.

¦ SCROLD, VAR%

Same as the scroll up routine except that the top line is cleared to the current paper ink and the bottom line will be lost. See example 3.

Example 1.

```
10 C%=0 : P%=0 : PRINT "C000 ";
20 FOR A=49152 TO 51200 : C%=C%+1
30 IF C%>8  THEN PRINT" ";:GOSUB 100:PRINT:PRINT HEX$(A,4);" ";
40 ¦ROMP,A,@P%:PRINT HEX$(P%,2);" ";:NEXT A
50 END
100 FOR B=A-8 TO A
105 ¦ROMP,B,@P%
106 IF P% >31 AND P%<127 THEN PRINT CHR$(P%);: ELSE PRINT ".";
110 NEXT B:C%=1
120 RETURN
```

Example 2.

```
10 CLS
30 LOCATE 15,12:PRINT"COLOURS"
40 FOR B=0 TO 26
50 ¦WAIT,10
60 INK 1,-B+26
70 INK 0,B
75 BORDER -B+26
80 NEXT B:LOCATE 15,14:PRINT"WAITING"
90 ¦WAIT,200:LOCATE 16,16:PRINT"¦RINK"
100 ¦RINK
110 ¦WAIT,200:GOTO 10
```

Example 3.

```
5 MODE 1
10 LOCATE 8,12:PRINT
"SCROLLING JUST SCROLLING"
11 ¦WAIT,100
20 ¦SCROLU,10
30 ¦SCROLD,20
40 ¦SCROLU,10
50 ¦WAIT,100
60 GOTO 20
```

Example 4.

```
10 REM Five second delay.
20 PRINT" WAITING 5 SECONDS"
30 ¦WAIT,500
40 PRINT "Finished"
```

Example 5.

```
10 LOAD"FAVORITE.PIC",&C000:REM Not a shaded dump sorry.
30 ¦EPDUMP
40 GOTO 40
50 REM You will have to use ESC to get out of this one.
```

¦ WAIT, VAR%

Wait for a period dependent on the value of the integer variable. A value of 100 will give a delay of approximately 1 second. The variable must *not* have a zero value, or very long waits will result. See example 4.

¦ EPDUMP

Gives a high resolution dump to an Epson compatible printer (sorry not DMP1). The routine will print anything from the screen that is not the current paper colour. Setting the paper colour prior to calling this routine may be used to print in inverse. If the printer is not on line, the routine will wait until it is. Escape is not possible unless the printer is on line. To escape when the printer is on line, hold down the escape key until BASIC is re-entered. The routine will return automatically when the transfer is complete. See example 5.

Example 6.

```
10 P%=0
12 FOR Y%=1 TO 25
15 FOR X%=1 TO 20
20 ¦COPYC,X%,Y%,@P%
30 LOCATE X%+20,Y%
40 PRINT CHR$(P%)
50 NEXT X%,Y%
```

¦ COPYC, X%, Y%, @VAR%

This routine returns the ASCII value of the character at the text screen coordinates X% and Y%. 664 and 6128 owners already have this routine progammed as part of BASIC.

Example 6 will copy the left side of the screen to the right side of a mode 1 screen.

¦ BASE, VAR%

The screen memory may be placed in any area of RAM but it must start on a 16K boundary. This routine will allow you to alter the start address of the screen RAM. The integer variable should be loaded with the value of the

most significant byte (MSB) of the 2-byte start address of the required screen, e.g. the default screen address is at &C000, and so the MSB is &C0. The only other useful address is at HEX 4000 (MSB &40). Be careful as other addresses may corrupt data. See example 7.

¦ OFFSET, VAR%

The screen base may be set on a 16k boundary. The offset routine allows the screen start address to be set in two byte increments offset from the screen base. The value of the integer variable may be in the range 0 to 255. See example 8.

¦ JOY1, @X%, @Y%, @T%
¦ JOY2, @X%, @Y%, @T%

The routines JOY1 & 2, when called, will update the X and Y coordinates depending on which direction the joystick is being pushed. It will also return the value '1' if the trigger is pressed.

All three variables must have a value before being called. (Normally the last screen coordinate.) If both joysticks are being used, different variable names should be assigned for each stick. With a joystick fitted, try example 9.

Example 7.

```
¦BASE,&40     [ENTER]
Type any characters you wish on the screen.
¦BASE,&C0     [ENTER]

Now type in this program.

10 ¦BASE,&40
20 ¦WAIT,100
30 ¦BASE,&C0
40 ¦WAIT,100
50 GOTO 10
```

Example 8.

```
20 CAT
30 FOR A%=1 TO 40
40 ¦OFFSET,A%
50 ¦WAIT,20
60 NEXT A%

Type in; MODE 1   [ENTER]
       ; LIST     [ENTER]
       ; RUN      [ENTER]
```

Example 9.

```
10 X%=0:Y%=0:T%=0
20 ¦JOY1,@X%,@Y%,@T%
30 PLOT X%,Y%,1
40 IF T%=1 THEN END
50 GOTO 20
```

ROM Programming

It is assumed that the programmer is conversant with machine code programming at least to the level of writing their own RSX programs in RAM. There are many good books that provide information on the operating system of the Amstrad 464, 664 and 6128, not least Soft 158 'The Complete CPC 464 Operating System Firmware Specification' published by Amsoft. This publication was found to be invaluable when preparing software for our range of Amstrad projects.

Although not absolutely necessary, a good 'assembler', capable at least of assembling to cassette or disk the machine code addressed from #C000 upwards, will be found most useful for the preparation of ROM based programs.

You will need access to an EPROM programmer and eraser. Ideally, if large amounts of code are to be put into ROM, the EPROM programmer should be able to take code directly from the memory of the Amstrad or from cassette or disk, and transfer it to the ROM. A suitable programmer will soon be available from Maplin.

ROM Format

There are three recognised types of ROM based programs:

Foreground
Extension
Background

An example of a Foreground ROM is the on board BASIC, and although it physically shares the same ROM with the operating system, the 16K bytes that contain the BASIC program are treated by the computer as a separate ROM. It is the responsibility of the Foreground ROM to take control of the computer, and to carry out all the house-keeping of the system.

If there is insufficient room in a Foreground ROM to contain the whole program, then additional ROMs can be added, and these are known as Extension ROMs.

To write a Foreground program requires a high level of competence in machine code programming, and a thorough understanding of the computer. It is not within the scope of this pamphlet to cover this particular aspect of external ROMs, and we will confine our efforts to demonstrating background ROMs only.

External ROMs overlay the default screen area of RAM starting at #C000, up to, depending on the size of ROM, #FFFF. The first few memory locations of an external ROM must follow a set pattern:

C000 ROM Type
C001 ROM Mark
C002 ROM Version
C003 ROM Modification

Of this block of four bytes, only the first is of any significance to the operating system – the ROM Type. The 'Mark', 'Version', and 'Modification' are chosen at the programmer's convenience.

*(C000)
Type #00 = Foreground ROM
Type #01 = Background ROM
Type #02 = Foreground Extension
Type #80 = On board BASIC (Foreground ROM with bit 7 set)
C004 Address of Command Table (low byte)
C005 Address of Command Table (high byte)
C006 Command Jump Block
C007 Command Jump Block
C009 Command Jump Block
C00A Command Jump Block

The length of the Command Jump Block will depend on the number of routines in the ROM that are to be called from BASIC.

The Command Jump Block consists of a number of calls to the routines programmed into the extension ROM.

Command Table
C00B Command Name
C00C Command Name
C00D Command Name
C00E Command Name
C00F #00

The length of the Command Table will depend on the number of routines in ROM that are to be called from BASIC, the null byte (#00) terminating the Command Table.

The Command Table is a list of the names of the routines that can be called from BASIC. The last byte of each name has the seventh bit set to indicate to the operating system that it is the last character of the name.

Each call in the Jump Block points to the location of its own associated machine code routine in the expansion ROM, and *not* to the commands in the Command Table, however, the calls must follow in the same order as in the list of commands. The Command Table is terminated in a null byte (00).

An assembler listing of the opening bytes of a Background ROM may resemble that shown in Listing 1.

DEFM, DEFB, etc., are assembler directives and may vary from assembler to assembler. Consult your assembler manual.

Passing Parameters

Parameters may be passed to an expansion ROM by the same method used by BASIC. The values are tagged onto the end of the command word each separated by a comma. BASIC will create a stack of the values

```
BC02                    70 RESINK: EQU  #BC02          ;Reset the screen pack
BB75                    80 SETCUR: EQU  #BB75          ;Set the cursor position
BB60                    90 GETCHA: EQU  #BB60          ;Read character at cursor
BC08                   100 SETBAS: EQU  #BC08          ;Set the screen base
BC0B                   110 GETBAS: EQU  #BC0B          ;Get the screen base and
B912                   120 GETROM: EQU  #B912          ;Get ROM address
                       130
C000                   140         ORG  #C000
                       150 ;
C000  01               160 START:  DEFB #01            ;Background ROM
C001  01               170         DEFB #01            ;Mark 1
C002  02               180         DEFB #02            ;Version 2
C003  03               190         DEFB #03            ;Modification 3
                       200 ;
C004  15C0             210 JBLOCK: DEFW COMND          ;Address of comnd table
C006  C334C0           220         JP   INIT           ;Jump block start
C009  C33EC0           230         JP   RINK
C00C  C342C0           240         JP   COPYC
C00F  C35BC0           250         JP   BASE
C012  C36BC0           260         JP   OLDBAS
                       270 ;
C015  4558414D         280 COMND:  DEFM "EXAMPLE RO"   ;Command table
C01F  CD               290         DEFB "M"+#80        ;Note! The last letter
C020  52494E           300         DEFM "RIN"          ;of each command name
C023  CB               310         DEFB "K"+#80        ;has bit 7 set to
C024  434F5059         320         DEFM "COPY"         ;indicate that it is the
C028  C3               330         DEFB "C"+#80        ;last.
C029  424153           340         DEFM "BAS"
C02C  C5               350         DEFB "E"+#80
C02D  4F4C4442         360         DEFM "OLDBA"
C032  D3               370         DEFB "S"+#80
C033  00               380         DEFB #00            ;Comnd table end marker
                       390 ;
```

Listing 1.

and point the IX register to the last entry. A point to note here is that each integer value on the stack will take up two bytes of the Amstrad's RAM (least significant byte first).

The ROM may retrieve values passed to it from BASIC by loading the Z80 registers with the contents of memory locations pointed to by the IX registers:

LD D, (IX+3) First entry on the stack
LD E, (IX+2)
LD H, (IX+1)
LD L, (IX+0) Last entry on the stack

NB. The alternate register set is used by the operating system and should not be used by the programmer.

A ROM may pass information back to BASIC via variables. The @ symbol, when placed in front of a variable, will return the address in memory where the variable has stored its current value.
For example:–

```
10  P%=0:PRINT @P%
```

The number printed on the screen is the address in RAM where the variable has stored the value zero, and *not* the actual stored value. A variable's address cannot be relied on to always be the same, as it is allocated dynamically.

A BASIC variable does not have an address until it has been given a value, any value will do, even '0'. A variable cannot be used for returning information from a machine code routine until it has an address.
For example:–

```
10  P%=0: POKE @P%,255: ?P%
```

In this example we have first set P% to zero (0), then POKEd the address of P% with the value 255, and then PRINTed P%. P% now holds the value 255.

As before, when a command is called BASIC creates a stack pointed to by the IX register, only this time if any of the variables have the @ symbol in front of them then the *address* of the variable is placed on the stack, and *not the value!* The ROM program can now use this address in which to place any information it may wish to send back to the BASIC program. This method of passing parameters via the IX registers is the same for RSX programs as well as external ROM programs.

Many ROM based programs will require an area of RAM to store run time variables etc., but because RAM space is allocated dynamically, it is not possible to reserve a few bytes and be sure that other ROM or RAM programs will not overwrite them. When a ROM routine is called, the IY register will be pointing to the bottom of an area of RAM that has been set aside by the ROM for its own use. We will see how this area is reserved in the next section.

If you already have a number of RAM RSXs that you intend to put into ROM, you will have to set up the first few bytes as shown earlier. Your next step will be to re-allocate the variable space. For each separate variable in the routine, you must replace the fixed address with an address pointed to by the IY register:

Variable 1 = (IY+0) Low byte
(IY+1) High byte
16-bit variable
Variable 2 = (IY+2) Low byte
(IY+3) High byte
16-bit variable
Variable 3 = (IY+4)
8-bit variable
Variable 4 = (IY+5)
8-bit variable

Initialisation Routine

Reserving RAM Space

When the Foreground program initialises the external ROMs, the first routine in each ROM is entered and run. This is known as the initialisation routine and it is responsible for reserving RAM space, putting out start up messages, etc. The user should not be able to access this routine, and placing spaces in the command name is one method of prevention. (BASIC will not accept spaces in a command but the operating system will.)

When the initialisation routine is entered the following information is held in the Z80 registers.

DE contains the address of the lowest byte of RAM available.

HL contains the address of the highest byte of RAM available.

One of the first actions that the initialisation routine should take is to check the position of the ROM. This serves two purposes. One is to allow the ROM to provide information for setting up side calls (calling routines between ROMs), and the other is to allow for variations in the ROM logging routine in different operating systems. (The 464 tries to log in 8 ROMs whereas the 664 attempts to log in 16.) The operating system routine, KL CURR SELECTION #B912, when called from a ROM will return with the ROM select address in the A register. If the select address is greater than 7 then the initialisation routine should not reserve any RAM and clear the carry flag before returning control to the operating system. If the ROM select address is between 0 and 7

```
C034  B7       400 INIT:   OR   A              ;Clear the carry flag
C035  CD12B9   401         CALL GETROM        ;Check ROM Address
C038  CB5F     402         BIT  3,A           ;Greater than 8 ?
C03A  C0       403         RET  NZ            ;Return if it is
C03B  2B       404         DEC  HL            ;Reserve 1 byte RAM
C03C  37       405         SCF                ;Set the carry flag
C03D  C9       410         RET                ;Return to Op/Sys
               420 ;
```

Listing 2.

```
C03E  CD02BC   430 RINK:   CALL RESINK        ;Reset the screen pack
C041  C9       440         RET                ;also resets inks to
               450 ;                          ;default values
```

Listing 3.

```
C042  FE03     460 COPYC:  CP   3              ;Check for 3 parameters
C044  C0       470         RET  NZ            ;Return if not
C045  DD6601   480         LD   H,(IX+1)      ;Get return address HI
C048  DD6E00   490.        LD   L,(IX+0)      ;Get return address LO
C04B  E5       500         PUSH HL            ;Save it
C04C  DD6604   510         LD   H,(IX+4)      ;Get X coordinate
C04F  DD6E02   520         LD   L,(IX+2)      ;Get y coordinate
C052  CD75BB   530         CALL SETCUR        ;Set the cursor
C055  CD60BB   540         CALL GETCHA        ;Get the character
C058  E1       550         POP  HL            ;Get return address
C059  77       560         LD   (HL),A        ;Put the char in it
C05A  C9       570         RET                ;Return to caller
               580 ;
```

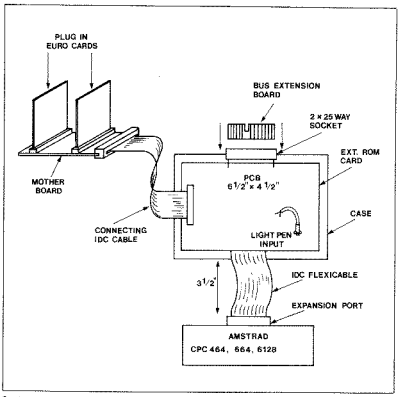
Listing 4.

the routine may go ahead and reserve RAM.

The block of memory between DE and HL will vary depending on the requirements of other external ROMs that may have been initialised before the current ROM. The initialisation routine may now reserve RAM space by modifying HL, DE, or both. To set aside say 5 bytes of RAM at the top of the memory pool, simply subtract 5 from the value of HL. To reserve memory at the lower end of the pool, add the required amount to DE.

If your initialisation routine is required to do things other than just reserving memory, then it may be necessary to push HL and DE. It is the values contained in these registers, when the routine returns to the operating system, that are used to reserve memory. From now on, whenever a routine in this ROM is called, the IY registers will point to the address you gave the HL registers. Access to the lower area of memory, reserved by altering DE, is a little more difficult and involves the use of pointers set up in the upper area. It is not automatic.

If no RAM space or other action is required by the ROM on initialisation, the routine should still check its select address and set the carry flag before returning control to the operating system.



System overview

Working Examples

Once the principles involved in the passing of parameters back and forth between BASIC and ROM routines are understood, there is very little difference between programming for RAM or ROM. The next few programming examples are designed to demonstrate the principles which up to now we have only talked about.

The initialising routine labelled INIT in Listing 2 is the first routine in the ROM and is entered and run automatically by the operating system on EMS (if the ROM is selected).

The first action the routine labelled INIT takes is to OR the A register, this has the effect of clearing the carry flag. A call is then made to the lower ROM routine that returns the ROM select address in the A register. Bit 3 of the A register is checked to ensure that the select address is not between 8 and 16, if it is then the routine returns control to the operating system. If the ROM select address is between 0 and 7 then the routine goes ahead and reserves one byte of RAM by decrementing the HL register pair, the carry flag is set and control is passed back to the operating system via the RET directive.

Listing 3 is probably one of the shortest ROM routines you will use. RINK calls the routine in the lower operating system ROM, via the system jump block, that resets the screen pack. One of the advantages of this routine is that the inks are all reset to their default values. Very useful if you accidentally blank out your screen while experimenting with the inks.

If the routine has been called by BASIC, then the return will be to BASIC, but a point to note here is that most routines in the ROM can be called by other routines in the same ROM, simply by a call to the routine's address. In these cases, the return will be to the calling routine.

Routines in one ROM may be called from another ROM, but the method used is known as a 'Sideways Call'. This is explained in Amsoft's Soft 158.

Listing 4 shows the first of the routines that actually passes parameters.

Many of you that have the 664 and 6128 will know this routine, it gets a character from the current cursor position on the screen and returns it in a variable to BASIC. This particular routine

```
C05B  FE01      590 BASE:   CP   1                ;Check one parameter
C05D  C0        600         RET  NZ              ;Return if not
C05E  CD0BBC    610         CALL GETBAS          ;Get the current base
C061  FD7700    620         LD   (IY+0),A        ;Save it in (IY+0)
C064  DD7E00    630         LD   A,(IX+0)        ;Get user base
C067  CD08BC    640         CALL SETBAS          ;Set it
C06A  C9        650         RET                  ;Return to caller
                660 ;
```

Listing 5.

```
C06B  FD7E00    670 OLDBAS: LD   A,(IY+0)        ;Load a with old base
C06E  CD08BC    680         CALL SETBAS          ;Set base with old
C071  C9        690         RET                  ;Return to caller
```

Listing 6.

requires 3 parameters and is called using the following command:

```
¦ COPYC,VAR%,VAR%,@VAR%
```

Where VAR% represents an integer variable and @VAR%, as explained earlier, represents the *address* of an integer variable.

When this command is called, BASIC creates a stack of the parameters, with the IX register pair pointing to the last entry. In this case, the last parameter to be put on the stack will be the address of the variable for the return of the character's value to BASIC. The address will be passed as a 16-bit number, so it must be held in two memory locations, the low byte being followed by the high byte.

The IX register pair will now be pointing to the low byte of the address, it therefore follows that IX+1 points to the high byte. The same principle applies to the other parameters. Each variable is a 16-bit number and is stored in two memory locations. This may be a little difficult to follow at first, but if you remember that most stacks are built from the top downwards, it makes sense that the last value added is the lowest in memory, and therefore IX+0 is pointing to the entry lowest in memory.

In the COPYC routine, the first action is to check that the correct number of parameters have been passed. Fortunately, when BASIC is creating the stack of parameters, it keeps a check of the number and before calling the routine, loads this value into the A register. If we know how many parameters there should be, a simple compare within the routine can check for the correct number and return if an error is detected.

The next action of the routine is to load the register pair HL with the address of the character returning variable, then save it on the system stack. The X and Y co-ordinates of the cursor are then loaded into HL. We appear to have skipped IX+3, but remember that the largest X co-ordinate value will not be greater than 80 in decimal, and the largest Y value not greater than 25, and each of these values will fit into one 8-bit byte. There is little point in defining the most significant byte as in each case, it will be zero.

H and L now contain the X and Y co-ordinates respectively and the system routine to set the cursor (#BB75) can now be called. The system routine to get a character from the screen at the current cursor position (#BB60) has no entry conditions, and can be called immediately after. On returning, the A register holds the ASCII value of the character found at the cursor co-ordinates. The address of the variable we are using to return the character to BASIC is then POPed off the system stack into HL, and the value in A register is then loaded into the address pointed to by HL. A return is then made to BASIC.

These next two routines demonstrate the use of the IY register pair, and use the byte of RAM that we reserved during the initialising routine.

Listing 5, BASE, when called by BASIC, will have one parameter, the MSB of the screen base address. The routine first checks to see that there is only one parameter and returns to BASIC if there is not. If all is well, a call is then made to the system routine that gets the current base address, and this is then stored as a variable in the byte of RAM we reserved during initialisation. A point to note here is that BASIC (the foreground program) sets the IX registers, but the operating system sets the IY registers.

Listing 6, OLDBAS, has been included purely as a demonstration and should not be used in programs of your own without providing some means of checking that IY+0 contains a valid screen base address. You could corrupt data areas if a value other than &C0 or &40 is passed to it.

The routine takes the MSB of the required screen base address from the location pointed to by the IY register pair plus the displacement, and loads it into the A register. In this demonstration, the routine BASE will have placed the MSB of the screen base address in (IY+0), and should have been called at least once before using OLDBAS.

The system routine to set the base is then called, and a return is made to BASIC.

Listing 7 is a compilation of the other assembler listings and is taken from a working ROM. You could, if you wish, use this as your first ROM program.

EXTERNAL ROM CARD PARTS LIST

RESISTORS All 0.6W 1% metal film unless specified			
R1	2k2	1	(M2K2)
R5-12	4k7 SIL	1	(RA29G)
CAPACITORS			
C1	100µF 10V PC Electrolytic	1	(FF10L)
C2-4	1µF 35V Tantalum	3	(WW60Q)
C5	100nF Minidisc	1	(YR75S)
C6	100µF 16V PC Minelec	1	(RA55K)
SEMICONDUCTORS			
D1	0A47	1	(QH70M)
IC1	74LS32	1	(YF21X)
IC2	74HCT137	1	(UB32K)
IC3	74HC30	1	(UB14Q)
MISCELLANEOUS			
	Controller PCB	1	(GD05F)
	Extendiboard PCB	1	(GB99H)
	14-way DIL socket	2	(BL18U)
	16-way DIL socket	1	(BL19V)
	28-way DIL socket	8	(BL21X)
S1	DIL switch SPST Octal	1	(XX27E)
	Box vero 201	1	(LL05F)
	Front panel	1	(FA88V)
	Back panel	1	(FA89W)
	Spacer 4BAx¼in.	1 Pkt	(FW31J)
	Self tap screws No 4x¼in.	1 Pkt	(BF66W)
	50-way Amstrad cable	1	(FA86T)
	2x25-way edgeconn	1	(FA87U)
	IDC polarising key	1	(QY73Q)
JK1	3.5mm Stereo jack skt	1	(FK03D)
PL1	3.5mm Stereo jack plg	1	(FJ99H)
	Amstrad pamphlet	1	(XH65V)
OPTIONAL			
	EPROM 2716		(QQ07H)
	EPROM 2732		(QQ08J)
	EPROM 2764		(QQ09K)
	EPROM 27128		(YH88V)
	Test ROM 2716/M11		(UF73Q)

MOTHERBOARD PARTS LIST

RESISTORS All 0.6W 1% metal film			
R2	47k	1	(M47K)
R3	10k	1	(M10K)
R4	4k7	1	(M4K7)
CAPACITORS			
C7	100µF 16V PC Minelec	1	(RA55K)
C8,9	100nF Minidisc	2	(YR75S)
C10	1µF 35V Tantalum	1	(WW60Q)
SEMICONDUCTORS			
IC4	74LS138	1	(YF53H)
IC5,7,8	74HC244	3	(UB65V)
IC6	74HC245	1	(UB67X)
MISCELLANEOUS			
	Motherboard PCB	1	(GD04E)
	2x25-way Transheader	1	(FT66W)
	IDC Polarising key	1	(QY73Q)
	16-way DIL socket	1	(BL19V)
	20-way DIL socket	4	(HQ77J)
	Veropins 2145	1 Pkt	(FL24B)
	Trackpins	1 Pkt	(FL82D)
OPTIONAL			
	64-way PCB rec		(FJ47B)

```
                    10 ;****************************
                    20 ;* EXTERNAL ROM PROGRAMMING *
                    30 ;* ----====== * ========== *
                    40 ;*      WORKING EXAMPLE      *
                    41 ;*    Version  110286/3      *
                    50 ;****************************
                    60 ;
BC02                70 RESINK: EQU  #BC02              ;Reset the screen pack
BB75                80 SETCUR: EQU  #BB75              ;Set the cursor position
BB60                90 GETCHA: EQU  #BB60              ;Read character at cursor
BC08               100 SETBAS: EQU  #BC08              ;Set the screen base
BC0B               110 GETBAS: EQU  #BC0B              ;Get the screen base and
B912               120 GETROM: EQU  #B912              ;Get ROM address
                   130
C000               140         ORG  #C000
                   150 ;
C000  01           160 START:  DEFB #01                ;Background ROM
C001  01           170         DEFB #01                ;Mark 1
C002  02           180         DEFB #02                ;Version 2
C003  03           190         DEFB #03                ;Modification 3
                   200 ;
C004  15C0         210 JBLOCK: DEFW COMND              ;Address of comnd table
C006  C334C0       220         JP   INIT               ;Jump block start
C009  C33EC0       230         JP   RINK
C00C  C342C0       240         JP   COPYC
C00F  C35BC0       250         JP   BASE
C012  C36BC0       260         JP   OLDBAS
                   270 ;
C015  4558414D     280 COMND:  DEFM "EXAMPLE RO"       ;Command table
C01F  CD           290         DEFB "M"+#80            ;Note! The last letter
C020  52494E       300         DEFM "RIN"              ;of each command name
C023  CB           310         DEFB "K"+#80            ;has bit 7 set to
C024  434F5059     320         DEFM "COPY"             ;indicate that it is the
C028  C3           330         DEFB "C"+#80            ;last.
C029  424153       340         DEFM "BAS"
C02C  C5           350         DEFB "E"+#80
C02D  4F4C4442     360         DEFM "OLDBA"
C032  D3           370         DEFB "S"+#80
C033  00           380         DEFB #00                ;Comnd table end marker
                   390 ;
C034  B7           400 INIT:   OR   A                  ;Clear the carry flag
C035  CD12B9       401         CALL GETROM             ;Check ROM Address
C038  CB5F         402         BIT  3,A                ;Greater than 8 ?
C03A  C0           403         RET  NZ                 ;Return if it is
C03B  2B           404         DEC  HL                 ;Reserve 1 byte RAM
C03C  37           405         SCF                     ;Set the carry flag
C03D  C9           410         RET                     ;Return to Op/Sys
                   420 ;
C03E  CD02BC       430 RINK:   CALL RESINK             ;Reset the screen pack
C041  C9           440         RET                     ;also resets inks to
                   450 ;                               ;default values
C042  FE03         460 COPYC:  CP   3                  ;Check for 3 parameters
C044  C0           470         RET  NZ                 ;Return if not
C045  DD6601       480         LD   H,(IX+1)           ;Get return address HI
C048  DD6E00       490         LD   L,(IX+0)           ;Get return address LO
C04B  E5           500         PUSH HL                 ;Save it
C04C  DD6604       510         LD   H,(IX+4)           ;Get X coordinate
C04F  DD6E02       520         LD   L,(IX+2)           ;Get y coordinate
C052  CD75BB       530         CALL SETCUR             ;Set the cursor
C055  CD60BB       540         CALL GETCHA             ;Get the character
C058  E1           550         POP  HL                 ;Get return address
C059  77           560         LD   (HL),A             ;Put the char in it
C05A  C9           570         RET                     ;Return to caller
                   580 ;
C05B  FE01         590 BASE:   CP   1                  ;Check one parameter
C05D  C0           600         RET  NZ                 ;Return if not
C05E  CD0BBC       610         CALL GETBAS             ;Get the current base
C061  FD7700       620         LD   (IY+0),A           ;Save it in (IY+0)
C064  DD7E00       630         LD   A,(IX+0)           ;Get user base
C067  CD08BC       640         CALL SETBAS             ;Set it
C06A  C9           650         RET                     ;Return to caller
                   660 ;
C06B  FD7E00       670 OLDBAS: LD   A,(IY+0)           ;Load a with old base
C06E  CD08BC       680         CALL SETBAS             ;Set base with old
C071  C9           690         RET                     ;Return to caller
```

Listing 7.

AMSTRAD ROM CARD 9 May 1986

CORRIGENDUM

AMSTRAD EXPANSION SYSTEM PAMPHLET XH65V

Page 9 Sub heading "TESTING"

Paragraph 1
" Do not fit any ICs at this stage"

Should read;
" Do not fit any ICs except IC3"

Paragraph 3
" Switch off power and insert IC1 (74LS32), IC2 (74HCT137) and IC3 (74HC30)."

Should read;
" Switch off power and insert IC1 (74LS32) and IC2 (74HCT137)."

MOD 1

If the Amstrad is subject to crashing during initialisation

The ROMDIS PCB track between the cathode of D1 and the I/O expansion bus should be disconnected by cutting the track at the point marked on figure 1 below. None of our future expansion project will require this line and it is advisable to carry the modification as a matter of course.

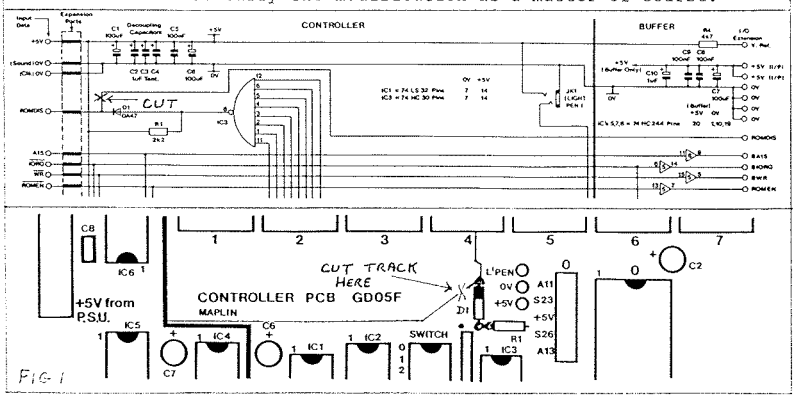


FIG 1