MASTER 128
SYSTEM ROM
USER GUIDE

Acorn
The choice of experience.

Copyright © Acorn Computers Limited October 1989
Designed and written by Acorn Computers Technical Publications Department

Neither the whole nor any part of the information contained in, or the product described in, this Guide may be adapted or reproduced in any material form except with the prior written approval of Acorn Computers Limited.

The product described in this Guide and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this Guide) are given by Acorn Computers Limited in good faith. However, Acorn Computers Limited cannot accept any liability for any loss or damage arising from the use of any information or particulars in this Guide.

The product described in this Guide (Master 128 System ROM part number 0243,940) is suitable only for use with the BBC Master 128 computer (product code AMB 15). Use with any other Acorn computer products is prohibitted.

Within this Guide, the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

All correspondence should be addressed to:

Customer Services
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

Information can also be obtained from the Acorn Support Information Database.(SID). There is a direct dial viewdata system available to registered SID users. Initially, access SID on (0223) 243642: this will allow you to inspect the system and use a response frame for registration.

ACORN, ACORNSOFT, MASTER, MASTER COMPACT, THE TUBE, VIEW and VIEWSHEET are trademarks of Acorn Computers Limited.

Published by Acorn Computers Limited

Issue 1
Part number 0443,941

MASTER 128 SYSTEM ROM USER GUIDE

Introduction

The alternative Master 128 System ROM incorporates a number of additional features beyond the standard production version. These include:

- extra commands
- extensions to existing commands
- a relocator to use with programs that are run on a 6502 Second Processor
- increased speed in some areas
- extended 8-bit character handling for foreign language use.

This leaflet only describes the changes in the alternative ROM, when compared with the standard one. It is split into four sections. The rest of this section details how to install the ROM in your computer, and what the new version numbers of the software are. The second section describes the changes that will affect all users; the third contains those changes most likely to affect only programmers. Finally an appendix gives details of how to write code for use with the relocator.

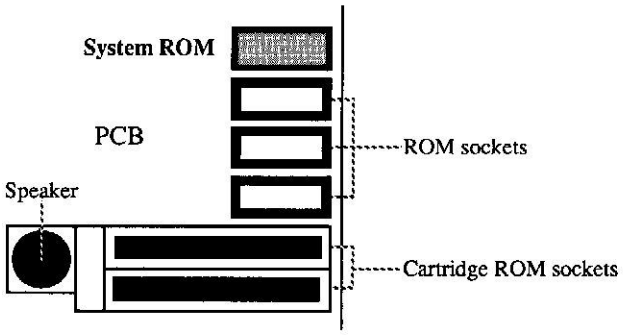
It is important to note that although the alternative ROM is highly compatible with existing application software it is not 100% compatible due, in particular, to the extended 8-bit character handling. For this reason the ROM is provided as a customer upgrade option, rather than being provided as standard in the computer. If you have any doubt about a particular piece of software, experiment with it before putting it to any serious use.

Fitting instructions

1 Disconnect the computer from the mains supply

2 Remove the computer's top cover by unscrewing the four screws labelled FIX (located on the computer's underside)

3 Position the computer so you are facing the keyboard, looking down on the main circuit board. Locate the standard system ROM, which is on the right hand edge of the board, towards the rear of the computer:



The socket is labelled IC24 on the circuit board.

4 Carefully remove the standard ROM, taking care not to bend the chip's legs, nor to damage the socket, nor to scratch the circuit board underneath it.

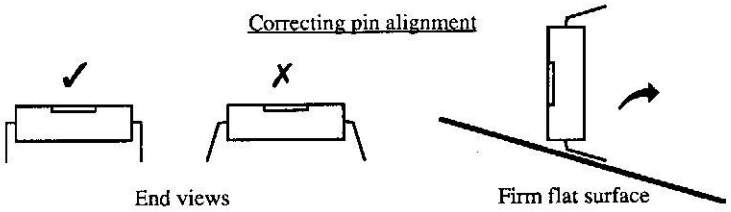
You should preferably use an IC extraction tool. If you do not have one available, insert a flat bladed screwdriver between one end of the chip and its socket. Gently lever the chip upwards **by a small amount only**, Then lever up the other end of the chip. Keep alternating between the two ends until the chip is released from the socket, then pull it out by hand.

A few Master 128 systems have the ROM soldered in. If this is the case, please contact your Acorn dealer.

5 Store the standard system ROM somewhere safe in case you want to use it again.

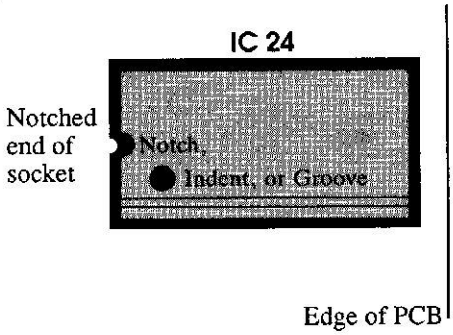
6 Touch an 'earthed surface' to discharge any static electricity, so you do not damage the alternative system ROM.

7 Pick up the alternative ROM by its plastic case between finger and thumb. Avoid touching the metallic pins. Check that all pins on the ROM are parallel and straight. If any appear crooked or splayed, you will need to realign them. Hold the chip sideways on and press the pins on one side gently against a firm flat surface:



Straighten the other row of pins if necessary.

8 Place the alternative ROM on top of the socket, so that the notch, indent or groove on the chip is positioned as shown below :



Make sure that all the legs of the chip have begun to locate in the holes of the socket. Apply firm pressure to the chip until you feel it press home, but do not force it. When the chip is in place, it appears to be slightly raised.

9 Check that all the pins have entered the socket and that none are bent or caught underneath.

10 Replace the top of the computer, and do up the four screws you removed earlier.

Version numbers

The changed software has new version numbers:

	Standard production ROM	Alternative ROM
MOS	3.20	3.50
Terminal	1.20	1.20
View	B3.0	B3.3
ADFS	1.50	2.03
BASIC	4	4r32
Edit	1.10	1.50r
Viewsheet	B1.0	B1.01
DFS	2.24	2.45

All the above code © Copyright Acorn except View and Viewsheet © Copyright Acornsoft.

USER DOCUMENTATION

Changes to the MOS

Support has been introduced for characters with the top bit set. See the later *Technical Documentation* for full details.

The `*BUILD` and `*APPEND` commands accept characters which have the top bit set: ie those with an ASCII value above 127.

The command `*SHOW` displays the values of all 16 soft keys (the function keys, the Break key, the cursor keys and the Copy key). They are numbered from 0 to F (hexadecimal). The standard ROM only shows the function keys 0 to 9.

You can generate a NUL character (ASCII code 0) by holding down the CTRL key and pressing the 0 (zero) key (Ctrl-0). You can still generate a NUL using Ctrl-@.

Keyboard translation is passed through the international keyboard ROM (if present) so that alternative character sets and keyboard layouts can be supported.

The `*UNPLUG` command unplugs all *identical* copies of a ROM. It is not necessary to issue this call once for each duplicate ROM.

The command `*REMOVE` *`file1 file2`* is rejected to avoid any possible confusion with the `*RENAME` command. This is especially likely when using abbreviations, for example `*RE.` *`file1 file2`* . (With the standard ROM, this command `REMOVE`s *`file1`*, whereas you might expect it to `RENAME` *`file1`* as *`file2`*.)

The command `OSRDCH` , when reading from the RS423 port in 8-bit transparent mode, correctly reads the character NUL.

The Ignore status (the character ignored by the printer) no longer alters the configured baud rate under certain circumstances.

The default configuration settings of the CMOS RAM have been changed:

- There is no need to change the configuration of your computer when installing the alternative ROM - their settings will not change. (The new default settings will only take effect if you ever have to reset the CMOS RAM, using an 'R power-on'.)
- If you are using an Econet network, you should make sure that both your configured fileserver and printserver have a network number specified as well. If you are running multiple networks, you will already have done this. If you are running a single network you should use `*CO. FS 0.254` rather than `*CO. FS 254` , which will give unpredictable results; likewise you should use `*CO. PS 0.235` .

Some characters in the font have been altered in appearance for better visual presentation.

The sideways RAM utilities (`*SRLOAD` etc) are now included in the MOS image.

With the standard ROM, when a bank of sideways RAM has a ROM image loaded into it using `*SRLOAD` , the command `*SRDATA` then generates the message `RAM occupied` . By popular request, this feature of the standard MOS has been removed from the alternative ROM.

If you are an Econet network user, and use the `*REMOTE` command to take over another computer, it is recommended that either both computers have the standard MOS, or that both have the alternative MOS. This will prevent any problems caused by different handling of ASCII NULs.

You are strongly advised not to start file names with a hyphen '-'. This prevents confusion with syntax used to specify file systems, such as `-net-` .

Osbyte 68 (Test RAM presence) has been improved so that it detects link changes without a hard break.

Note: Pages F.5-4 and F.5-5 of the *Master Series Reference Manual* are incorrect: LK18 always affects slots 4 and 5, and LK19 always affects slots 6 and 7.

An extra facility has been added to make a 6502 Second Processor easier and more efficient to use.

To get the most efficient use of the extra memory available in the Second Processor, it has been necessary to have a second version of the same ROM-based program (for example, BASIC and HIBASIC). The inconvenience of this has meant that, with the standard ROM, the same program is often used in either processor; thus not all the Second Processor's memory is used.

The alternative MOS supports a new type of ROM-based program, which automatically runs at different addresses on the two different processors. Thus only one copy of the program is needed, but the Second Processor is used more efficiently.

The new versions of BASIC and EDIT are of this type.

Full details of how this works, and how to write such ROMS yourself, are in the appendix entitled *The relocator*.

You may unfortunately find that some software written by third parties no longer works with the alternative ROM. This is almost always because the software is accessing the MOS in ways neither documented nor supported by Acorn (such as writing directly to memory-mapped devices instead of using the `*FX` calls provided). The changes made to the MOS mean that some of this badly behaved software will no longer work.

If you have such problems with your own programs, you will need to rewrite them to use calls documented and supported by Acorn. They should then work using all versions of the MOS, including any future ones.

Changes to Terminal

No changes have been made to Terminal, and its version number is unchanged.

Changes to View

A new command has been added so you can set how View starts:

```
*CONFIGURE VIEW SETUP [F],[J],[I]
```

The F, J and I are optional; if they are included:

F Format mode is on when View starts (an F will appear at the top left of the screen)

J Justify mode is on when View starts (a J will appear at the top left of the screen)

I Insert mode is on when View starts (an I will appear at the top left of the screen)

So `*CONFIGURE VIEW SETUP F` will start View so it is in format mode, but in neither justify mode nor insert mode.

View is configured by default to use CMOS slot 44 (decimal) and start in FJ state.

The new version of View handles characters whose top bit is set. The ASCII characters 128-134 inclusive are reserved for use by View and cannot be entered from the keyboard. Control characters can no longer be entered from the keyboard.

View still relocates itself if it is downloaded to a 6502 Second Processor.

An Epson printer driver is now incuded in the ROM and is selected as the default:

- to use the old default driver, type `PRINTER` while you are on the command screen
- to return to the Epson driver, type `PRINTER EPSON` on the command screen
- to load a printer driver called `EPSON` from the current filing system, include a drive or directory explicitly, for example `PRINTER :0.EPSON` .

When using the new printer driver, note that it:

- prints Highlight1 as underlined characters
- prints Highlight2 as bold characters
- does not generate line feeds
- passes characters whose top bit is set to the printer

Linefeeds used to be sent to the printer only, and hence could not be switched using `*FX 6` . Line feeds (if output) are now sent to the screen as well, so can be switched

using *FX 6 . However, this means that your screen may become blank when you are printing.

If you wish to use the extended highlight sequences, you must include the sequence HT 2 130 in your document.

Changes to ADFS

There is an improvement in speed when accessing floppy drives.

The *BACKUP command has been added. It copies an ADFS disc:

*BACKUP <source drive> <destination drive> S|M|L

The final parameter shows the type of disc to be used:

S 40 track single sided

M 80 track single sided

L 80 track double sided

The disc identifier bytes are not copied so that the copy has a separate identity to the source.

*COMPACT no longer needs any arguments, and the help text has been changed accordingly. It now uses filing system workspace, rather than using ordinary workspace

*BACKUP , *COMPACT and *COPY may all use the shadow screen memory as workspace if there is not enough free in the static workspace. If they do so, they will first attempt to select mode 135; if unsuccessful, the error message Bad MODE is given, indicating there is not enough free workspace to run the commands.

The *DRIVE command has been added to maintain compatibility with DFS programs:

```
*DRIVE <drive>
```

It is identical in effect to typing *DIR :<drive> .

The <drive> parameter can lie in the range 0-3; values of 2 and 3 are mapped to ADFS drives 0 and 1 respectively. Only the first character of the <drive> parameter is validated so that commands such as *DRIVE 0 80 can be entered.

The *FORMAT command has been added. It formats a disc:

```
*FORMAT <drive> S|M|L
```

The final parameter shows the type of disc to be formatted:

S 40 track single sided

M 80 track single sided

L 80 track double sided

The *VERIFY command has been added. It verifies an ADFS disc:

```
*VERIFY [<drive>]
```

If no drive is specified, the disc in the currently selected drive is verified.

A wider range of step values are supported. See the later *Technical Documentation.*

Changes to Edit

The syntax of the *EDIT command is no longer so rigidly checked. Both `*EDI.MYFILE` and `*EDIT  MYFILE` are now recognised.

EDIT now responds to the `*HELP` command with a full version number.

The new version of EDIT automatically runs at a different address if it is downloaded to a 6502 Second Processor. This makes the most efficient use of the extra memory that is available.

The new version of EDIT handles characters whose top bit is set.

EDIT can now handle both carriage returns and line feeds to signify the end of a line:

- an inverse '$' is now used to show a newline (either carriage return or line feed)
- an extra command, Ctrl-F8, has been added to change carriage returns to line feeds, and vice versa; this allows documents to be handled irrespective of which character they use to signify the end of a line.

The handling of non-breaking spaces has changed slightly as a result of the above:

- when in 'carriage return' mode, a non-breaking space can still be forced by using Ctrl-J; when in 'line feed' mode, a non-breaking space now has to be generated by using Ctrl-M, as Ctrl-J represents a line feed and so will force a new line
- a non-breaking space is no longer stored as ASCII 10 (Ctrl-J or line feed), but as ASCII 160
- the print formatter now converts all occurences of ASCII 160 to spaces.

You cannot load a new file without confirmation if you have already modified the current file since loading it.

EDIT's status display now shows the tab state.

EDIT also has an extended set of responses to the Shift-F9 (Clear Text) command:

- Esc (or waiting for a few seconds) will cancel the command
- Shift-F9 (a second time) will clear all text, and set the type of the file to Exec
- D will mark the text as discarded; it will not be deleted, but any subsequent command that would overwrite the text will do so without prompting you
- Y (or any other key) will clear all text.

Inserting a file no longer changes the current filename.

The new mode command now GETs the mode number rather than INPUTting it.

The find and replace command has an E(nd of file replace) option added. Also, it no longer gives occasional spurious matches at the end of a file.

The print command has an As Is option added, which prints the text without any formatting.

Changes to Viewsheet

Viewsheet now handles characters whose top bit is set. In the standard ROM the top bit of characters is used by the Justify Label command (Shift-F8), so as a result:

- the Justify Label command is no longer available
- if you have used this command on any fields of a spreadsheet, some of the characters in the field will have their top bit set, and so will be displayed by the alternative ROM as part of the extended character set (such as Greek letters).

Changes to DFS

You can now save files longer than 64k bytes.

All four step rates are now used.

`OSGBPB` now executes more quickly.

`CLOSE#0` leaves files with the correct length.

Changes to BASIC

The new version of BASIC automatically runs at a different address if it is downloaded to a 6502 Second Processor as the configured language when you reset your Master. This makes the most efficient use of the extra memory that is available.

It loads at its normal address if you load it at any other time (using the `*BASIC` command). To get it to load at the higher address, you must type `*HIBASIC` instead.

Various trigonometric and logarithmic functions have been recoded and now execute more quickly.

TECHNICAL DOCUMENTATION

Changes to the MOS

Support has been introduced for characters with the top bit set. Compatibility with earlier versions of the MOS has been maintained wherever possible:

- keys generating codes in the range &80 to &FF (such as the function keys) are handled as by the standard ROM: they are placed unaltered in the keyboard buffer, but when read by say OSRDCH they are prefixed by an ASCII NUL
- characters representing the extended ISO character sets (principally used to support languages other than English) are handled in exactly the opposite way: they are prefixed by an ASCII NUL when placed in the keyboard buffer, but when they are read from the buffer the ASCII NUL is stripped, so a single byte in the range &80 to &FF is returned.

The commands *FX221 to *FX 228 inclusive (which set the interpretation of input codes &80 to &FF) have also been extended. If a parameter of 2 is used (for example `*FX225,2`), then the code is prefixed by an ASCII NUL.

The COPY key now uses the *FX 135 command to read the character at the cursor position. It searches through the character definitions from ASCII 32 up to ASCII 127, then from ASCII 255 down to ASCII 128. When it finds a character that matches the one on the screen, the search stops and that character is copied. This order is used so that, if you have defined a character that happens to be identical to an international character, it is the international character that is matched and copied.

When you define characters you are advised to use ASCII 128 to ASCII 159 inclusive before using any other characters.

Code that intercepts a call to the MOS can cause the system to crash if it alters which ROM is selected, or what RAM is paged in. When writing such code you should ensure that the system is in the same state on exit as on entry. Two changes have been made to reduce the chances of such crashes:

- the sideways MOS is now reselected after a `BREAK` interception; this guards against interception code that leaves the sideways state undefined
- the FS RAM is now paged out on return from an interception of the printer or `VDU` indirections; this guards against calls made by the interception code to the MOS that leave it paged in.

An OSGBPB call (get/put multiple bytes) to the CFS (Cassette Filing System) now works correctly.

The *BASIC command used to directly enter BASIC. This is in contrast to all other languages, which issue a *FX142 to warn that the language is about to change. The *BASIC command now also issues a *FX142 for consistency.

Version messages now refer to the operating system as MOS rather than OS.

The drawing of long thin ellipses now works correctly.

A bug has been fixed in the standard ROM which leads to the corruption of a location called TEMPA. This happens if interrupts are enabled when sideways ROMs process an IRQ that is unknown to the MOS.

You may experience problems with any programs that illegally access the MOS directly. The best cure for such a problem is to rewrite the program so that it uses the facilities provided, documented and supported by Acorn. These are some of the changes that may be causing you problems:

- The *CONFIGURE and *STATUS commands are now table driven, not code driven.
- The OSBYTE and OSWORD jump tables have been moved to the sideways MOS
- The *BACKUP, *FORMAT and *VERIFY commands are now decoded by the MOS, rather than the filing systems
- The sideways RAM utilities (*SRLOAD etc) are now included in the MOS image
- The area move and ellipse code has been moved into the DFS ROM
- The Sound code has been moved into the DFS ROM.

These changes are listed for your information only, and are subject to further change in any future versions of the MOS.

Changes to Terminal

No changes have been made to Terminal, and its version number is unchanged.

Changes to View

Certain macro definitions are inaccessible during print operations when using the standard ROM. This has been fixed.

With the standard version of View, a list of filenames in a PRINT, SCREEN or SHEETS command is lost, when printing a file with a |D or |T reference in it. This is then followed by a Bad filename error. This has been fixed.

The L number register is now updated when printing a document with microspacing switched on.

The screen display is now correctly updated if you do a replace with a Ctrl-C in the result string.

Sideways RAM protection has been removed from View.

Changes to ADFS

The first read/write of a multi-sector operation now does a head settle. If the operation ends on the last sector of a 40 track drive it no longer causes a controller error.

`*COPY` no longer corrupts the user program space.

Write precompensation is now handled automatically, and no longer needs to be configured using `*CONFIGURE FDRIVE <value>` . The opportunity has been taken to widen the range of step values available, so that older disc drives needing a slow step rate can be used with the newer 1772 disc drive controller. The meaning of `<value>` depends on the disc controller fitted:

With a 1770:

0 a 6ms hardware delay

1 a 12ms hardware delay

2 a 20ms hardware delay and a 30ms software delay (50ms total)

3 a 30ms hardware delay.

With a 1772:

0 a 6ms hardware delay

1 a 12ms hardware delay

2 a 2ms hardware delay and a 30ms software delay (32ms total)

3 a 3ms hardware delay.

The handling of head settle delays has been improved. Consequently a skew value of 4 sectors is now used when formatting discs.

`OSGBPB` now returns a zero byte in the bytestream after the CSD and library names. This matches the ownership byte returned by the network, and has been introduced for compatibility between the two filing systems.

Changes to Edit

The special characters used to draw boxes around help text (for instance in D mode) are now explicitly defined within EDIT. When you leave EDIT, the old definitions of these characters will be restored.

The help text (as displayed in D mode) has been slightly altered for greater clarity.

Wiping to the end of a line is done by clearing a text block, instead of clearing a text widow as the existing ROM does. This no longer affects the cursor position in cursor editing mode.

When you leave EDIT, the function keys are now restored to their default states.

Changes to ViewSheet

All changes to ViewSheet are included in the *User Documentation* section.

Changes to DFS

Loading a file using OsFile &FF now returns A=1.

A disc catalogue buffered in memory is now marked as invalid after a soft reset. This safeguards against errors caused by changing discs immediately before a soft reset.

OSWord calls are now only accepted by the DFS if it is the current filing system.

A spurious motor-on no longer occurs after a 1770 reset.

Writing to the extent works correctly.

Changes to BASIC

A sideways ROM header has been added.

APPENDIX - THE RELOCATOR

Overview

The Second Processor has more usable memory than the main I/O processor. If a ROM-based program is to make use of this extra memory, it must be loaded at a higher address in the Second Processor (say &B800) to that it uses in the main I/O processor (typically &8000). This in turn means that the program itself must be different. With the existing MOS two approaches were used:

- to have a different version of the same program for the two processors (for example, BASIC and HIBASIC)
- to use the same program on either processor, thus not using the extra memory of the Second Processor.

The new MOS approach is:

- to support a new type of ROM-based program which can be run at a different address (or *relocated*) depending on which processor it is being run on.

When such a ROM-based program is downloaded through the TUBE to the Second Processor:

- it is automatically rewritten using information held in the ROM
- it is then ready to run at the new address, and is loaded there.

Technical details

If a program is assembled twice, using a different load address each time, the majority of bytes will be identical between the two versions. This includes bytes that specify:

- the type of instruction to be performed
- any relative addressing, since the target will still be the same number of bytes away
- text strings
- immediate data.

The only bytes that are likely to be different are those that specify an absolute address within the program. These are likely to change by a fixed amount, as all the code should have moved in memory by the same offset. If the two programs were ROM images, to be loaded at &8000 and &B800 respectively, you would expect such bytes to differ by &38 - the number of pages offset between the two load addresses.

A ROM image for the second processor can be derived from one for the main I/O processor if we know:

- which bytes differ from the corresponding ones in the other ROM image
- what the fixed offset is between such bytes.

A bit-map is used to keep track of which bytes need the offset added. Since the ROM itself only uses memory from pages &7F to &BF inclusive, all the bytes that do differ should lie in this range. Space is saved in the bit-map by only keeping a bit record for these bytes.

The relocator is an extension to the TUBE code. When a ROM is downloaded through the TUBE the relocator checks if:

- the Second Processor is a 6502
- the ROM is one of the new relocatable ones
- the bit-map with the ROM has the expected format.

If any of the above conditions are not met, the ROM is downloaded unaltered, and the relocator has no effect. If all the above conditions are met, the relocator selectively adds the offset to each byte as it is downloaded through the TUBE:

- Bytes from &00 to &7E have no corresponding bit kept in the bit map. These bytes are not altered by the relocator as they are downloaded.
- Bytes from &7F to &BF have a single corresponding bit kept in the bit-map. If the bit is set, the relocator adds the offset to the byte as it is downloaded; if the bit is clear, the byte is not altered
- Bytes from &C0 to &FF have no corresponding bit kept in the bit map. These bytes are not altered by the relocator as they are downloaded.

Limitations

The relocator will only work correctly if the bytes in the two ROM images that differ do so by a fixed amount. This implies that the relocation must be by a whole number of pages. (If the relocation were by an offset of &1234, then some bytes would differ by &12, and some by &34).

Avoiding relocation

The `*FX 142` call has been extended so you can choose whether or not a ROM image relocates as it is downloaded:

- if you want the ROM image to relocate, type `*FX 142 <ROM no.>`
- if you do not want the ROM image to relocate, type
  `*FX 142 <ROM no. + 64(decimal)>`

Writing a relocatable ROM

A relocatable ROM is very similar to any other ROM written to be downloaded through the TUBE; the only major addition is the bit-map. The ROMs' format is shown by the diagram - please refer to it as you read this section. You may also find the section of the *Master Series Reference Manual* entitled *F.7 Paged ROMS* useful.

Note that the order of the title, version number and copyright strings shown in the diagram is correct; the *Master Series Reference Manual* incorrectly states that the optional version number string follows the copyright string.

Note the following:

- bytes &00 - &02 must not be zero, since the ROM will contain a language
- the ROM must support a service call entry

In the ROM type byte (offset &06):

- the service entry bit (bit 7) must be set
- the language bit (bit 6) must be set, since the ROM contains a language
- the relocation bit (bit 5) must be set
- bit 4 must be clear
- the code type bits (bits 3-0) must specify 6502 code (0010)

All ROMs that are downloaded through the TUBE have a relocation address, which follows the copyright string. This is the address at which the ROM is loaded in the second processor. The relocator can differ between an old-type ROM (for use only in the Second Processor) and one of the new relocatable ROMs (for use in either processor) because:

- in an old-type ROM, the relocation address is followed by two null bytes
- in a relocatable ROM, the relocation address is followed by a pointer to the bit-map descriptor table.

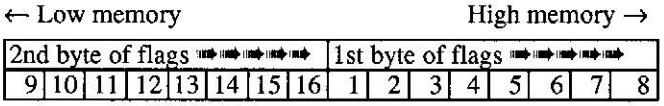
The bit-map descriptor table must lie in the same ROM as the header. It tells the relocator where to find the bit-map. The bit-map itself can lie in another ROM. The descriptor table gives the address of the byte **after** the last byte of the bit-map. It also specifies which ROM the bit-map is in:

- if the top-bit of the ROM number byte is set, the number is given relative to the ROM containing the descriptor table (so 10000000 would specify the same ROM as the descriptor table, 10000001 would specify the next highest ROM, and so on)
- if the top-bit of the ROM number byte is clear, the ROM number is absolute, and so the ROM containing the bit-map must be in a specific socket.

You should preferably use relative addressing; multi-chip sets should lie in adjacent sockets.

The bit-map is the greatest difference between an old-type ROM and a relocatable ROM. Note the following:

- the bit-map is filled from high memory to low memory
- each byte of the bit-map is filled from bit 7 to bit 0
- the last byte in the bit-map to be filled (labelled +0 in the later diagram) has any unused low-order bits padded with zeroes
- the two bytes `&C0DE` must be present as a redundancy check.



Order in which bit flags are filled (first two bytes shown).

A program is given at the end of this appendix which will automatically generate a bit-map for you. Before running the program you will need to assemble your ROM image twice: once to run at &8000; and again to run at the relocation address, which should be a higher address at the start of a page (the low byte of the address must be &00). Store the two images in separate files. When you run the program, it will prompt you for the two filenames, generate the bit-map, and save it in the file `bitmap`. Three error messages are possible:

`Could not open <filename>`

The program could not open the given file. Check you typed its name correctly, and are in the correct directory.

`Fatal error - bitmap table underflow`

The space allowed by the program for the bitmap (64k bits) was inadequate. Check you have correctly typed the program. If necessary, increase the space allocated to `bmbuff%` in line 80.

`Fatal error - difference not constant`

The program found a pair of bytes whose difference was not the same as that of previous ones. The address of the offending pair is also given, relative to the start of the ROM image. Look at your source code and correct any reason for this.

Relocatable ROM header

&00 - &02	JMP ROM language entry address
&03 - &05	JMP ROM service call entry
&06	ROM type flags - must be 11100010
&07	offset to start of copyright string
&08	version number in binary I ASCII NUL
&09...	title string
	ASCII NUL
	[version number string]
	ASCII NUL
	ASCII '('
	ASCII 'C'
	ASCII ')'
	rest of copyright string
	ASCII NUL
	relocation address: low byte
	relocation address: high byte
	pointer to bit-map descriptor table: low byte
	pointer to bit-map descriptor table: high byte

Bit-map descriptor table (must be in the relocatable ROM)

	pointer to byte after bit-map: low byte
	pointer to byte after bit-map: high byte
	relative/absolute ROM number bit-map is held in
	ASCII NUL (reserved for future use)

Bit-map (can be in another ROM if necessary)

+0	*nth* byte of flags (for bytes at end of ROM image)
+1	*(n-1)th* byte of flags
...	...
+(n-2)	2nd byte of flags
+(n-1)	1st byte of flags (for bytes at start of ROM image)
+n	length of bit-map *n:* low byte
+(n+1)	length of bit-map *n:* high byte
+(n+2)	&C0
+(n+3)	&DE
	This byte is pointed to by the descriptor table. **It is not part of the bit-map**

Program

```
  10 REM> GenBitMap
  20 REM generates bitmap for the given image
  30 REM the program requires two ROM images:
  40 REM the original (assembled at &8000)
  50 REM the alternative (assembled >= &8100) used to check for differences
  60 REM the bitmap (if valid) is always placed in the binary image "bitmap"
  70 :
  80 DIM bmbuff% &2000 : REM 8K should be large enough for the bitmap
  90 bend%=bmbuff%+&2000
 100 bptr%=bend% : REM start at the end
 110 :
 120 code%=&DEC0 : REM bitmap identifier
 130 dist%=&0000
 140 numbits%=&0000
 150 address%=&8000-1
 160 :
 170 INPUT "File containing ROM image assembled at &8000 => "realrom$
 180 INPUT "File containing ROM image assembled for relocation => "altrom$
 190 :
 200 rr%=OPENIN(realrom$)
 210 IF rr%=0 THEN PRINT "Could not open ";realrom$:END
 220 :
 230 ar%=OPENIN(altrom$)
 240 IF ar%=0 THEN PRINT "Could not open ";altrom$:CLOSE#rr%:END
 250 :
 260 REM -- the bitmap is stored in reverse order, with the bitmap header
 270 REM -- information stored at the end
 280 REM -- the header contains a special identifier (CODE) and the length
 290 REM -- of the bitmap in bytes
 300 bptr%=bptr%-2 : REM step over ID
 310 bptr%?&00=code% MOD &0100
 320 bptr%?&01=code% DIV &0100
 330 bptr%=bptr%-2 : REM step over bit count
 340 bptr%?&00=&00
 350 bptr%?&01=&00
 360 bcadr%=bptr%     : REM but remember where we need to place it
 370 :
 380 REPEAT
 390  :
 400  bptr%=bptr%-1
 410  val%=0
 420  bc%=8
 430  :
 440  REPEAT
 450   :
 460   v1%=BGET#rr%
 470   v2%=BGET#ar%
 480   address%=address%+1
 490   :
 500   IF ((v1% >= &7F) AND (v1% <= &BF)) THEN PROCcheck
 510   :
 520  UNTIL (bc%=0) OR (EOF#rr% OR EOF#ar%)
 530  :
 540  REM check and see if the last byte in the bitmap table needs to be
 550  REM padded (this is required due to the lo-address byte being stored
 560  REM in the highest bit)
 570  IF (bc%=0) THEN PROCinsert ELSE IF (bc%>0 AND bc%<8) THEN FOR loop%=bc% TO
1 STEP -1:val%=(val%*2)+0:NEXT:PROCinsert
 580  IF bc%=8 THEN bptr%=bptr%+1
 590  :
 600 UNTIL (EOF#rr% OR EOF#ar%) OR (bptr%<bmbuff%)
 610 :
```

```
  620 IF bptr%<bmbuff% THEN PRINT "Fatal error - bitmap table underflow"
  630 :
  640 CLOSE#rr%
  650 CLOSE#ar%
  660 :
  670 bcadr%?&00=numbits% MOD &0100
  680 bcadr%?&01=numbits% DIV &0100
  690 :
  700 save$="SAVE bitmap "+STR$~bptr%+" "+STR$~bend%+" 0 0"
  710 PRINT"About to save <";save$;">"
  720 IF bptr%>=bmbuff% THEN OSCLI(save$)
  730 :
  740 END
  750 :
  760 DEFPROCinsert
  770 REM place a byte into the bitmap, and decrement pointers
  780 bptr%?&00=val%:numbits%=numbits%+1
  790 ENDPROC
  800 :
  810 DEFPROCcheck
  820 REM - check if a byte needs relocating
  830 REM   distance apart
  840 LOCAL d%
  850 IF v1%=v2% THEN PROCaddbit(0):ENDPROC
  860 REM they are different
  870 REM check they are always the same distance apart
  880 d%=v2%-v1%
  890 IF dist%<>0 AND d%<>dist% THEN PRINT"Fatal error - difference not constant"
ELSE IF dist%=0 THEN dist%=d%
  900 :
  910 PRINT "Diff at &";~address%
  920 :
  930 PROCaddbit(1)
  940 ENDPROC
  950 :
  960 DEFPROCaddbit(v%)
  970 REM - add the bit given into the structure
  980 REM   this is done by remembering the bit in the current byte
  990 REM   bc%  : global that holds the bit count within the current byte
 1000 REM   val% : global that holds the current byte
 1010 bc%=bc%-1
 1020 val%=(val%*2)+v%
 1030 ENDPROC
```