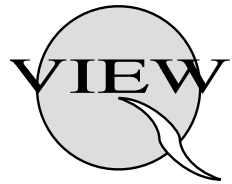
# Minerva

# Technical Guide

A differently-coloured cover, but just as much wisdom
from those awfully nice people at the




International MegaCorporation

## Assembler 49

# Introduction

History

First of all thank you for expressing your support for the QL and the many hours of work we have put into Minerva! Originally it was intended to be a pet project amongst ourselves but brief public exposure soon showed that there were other QL users who had also been wanting the same features as us. Being a friendly bunch of people we decided to give up our normal evening lives in pursuit of bringing the results of our labour of love to others less fortunate than ourselves!

So who are QVIEW anyway, and why are they affectionately known as the International MegaCorporation? These and other difficult questions will be answered in the following few paragraphs. If you already know or don't care then feel free to fast forward to the fitting instructions and play with your new toy; you can come back to us later.

QVIEW came into being a couple of years ago when two avid QL users who were into comms, modems and staying awake all night happened into each other. Laurence Reeves and I (Stuart McKnight) had been developing Viewdata Bulletin Board systems to run on a QL, each unaware of the other's activities. It was a chance phone call from Lau that brought us together and we decided to pool our efforts and produce a system which we hoped was better that the sum of the parts. About that time the mind bending QPTR Toolkit documentation from QJUMP was released, and somehow I found myself volunteering to write the page editor in QRAM style pull down windows while Lau got the difficult bits of getting the modem to answer the phone. What's so difficult about that you say? Well did you ever try to get an early Astracom modem to behave as it said it ought to (the current models are vastly improved thanks to Tony Price and Tony Firshman's spade work on the original code!).

A few other comms fanatics got to hear of our activities and wanted to run similar boards so our first non-commercial venture was born and other QL run bulletin boards appeared while we continued staying awake through the early hours of the morning either phoning around looking at them or writing updates to the software! Which is how we sort of roped Jonathan in as he was the QPTR expert, well if he didn't understand his own manual then what chance did I stand?! In true QVIEW lunatic tradition, Jonathan found himself idly staring at an LED, a transistor and a couple of resistors and invented the fiendish CAPSLED device to indicate when CAPSLOCK and Screen Freeze had been enabled. I liked it, decided I wanted one for my machine and then in a mad state of philanthropy wondered how many other people might like the additional LED in restful green. It was a rock-bottom price kit at £5.00 and we were pleased to know that despite our instructions, nobody had corpsed their machine as a result of fitting it.

So the QVIEW team are, in no particular order of importance.....

## Stuart McKnight

Stuart McKNIGHT, not actually a programmer or anything to do with computers in his daytime life, apart from the fact that they take up sizeable areas of his flat. A terminal Goon Show junkie he has by now seriously infected the other two with this manic form of humour as anybody who has heard us re-enacting certain favoured episodes will testify. He and Lau also share a sneaking admiration for A Very Peculiar Practice to which Jonathan as yet remains immune but we're working on it. Originally, dealings with QVIEW were via Stuart as he's the one who volunteered to answer the phone, open all the mail, sort it, answer it, answer the phone, mail out Minerva, answer the phone, organise visits to QUANTA workshops and by the way is the kettle on yet....yes, catering was also on his duty roster. He is currently trying to close the Terry Pratchett gap so that he can figure out why Lau and Jon fall about laughing for reasons he doesn't yet understand (something to do with strangely named camels performing complex mathematical calculations...?!!?!?!?!)

## Jonathan Oakley

Jonathan OAKLEY found his daytime activities of designing PCBs for odd little gizmos, chasing component suppliers and criticizing other people's software somewhat dull, so he joined forces with the Mega Corp and now spends his evenings and weekends designing odd little PCBs, chasing components and criticizing other people's software. His other main activity is visiting Stuart for Sunday lunch in an attempt to gain entry to the Guinness Book of Records for having eaten the same meal every Sunday lunchtime for a year or two....(or should I change the menu Jon?)(yes Stu you should: by the way, the phone's ringing. And where's the tea? − Jon)

## Laurence Reeves

Laurence (Lau) REEVES a.k.a The Grand Wizard is usually to be found amid a pile of ashtrays of incinerated Gauloise Disque Bleu cigarettes, deep in thought at the keyboard, in total command of the three hundred odd Minerva source files on the Miraculous Winnie looking for some new devious twist to add to an existing QDOS routine, to make it wondrously useful, totally incomprehensible, or both (wait 'till you see the new MODE command!). He claims most of his best ideas happen when he's in the bath, or waking up; we suspect that some of his real brainsmashers have occurred while he was doing both (but not whistling)! Other known hobbies include eating *calzone*, making sense of The Prisoner and counting the nuns. His one luxury marooned on a desert island would be a video of the film "Dark Star".

The Minerva operating system, named after the Roman Goddess of Wisdom, has been a year in the making. Our intention was to extend the possibilities of an already fine operating system and cure some of the more dangerous bugs that have come to light in the five or six years since QDOS first appeared. Originally produced purely for our own amusement, we were eventually persuaded to share it with fellow QL users.

In our travels through other people's software we have discovered an occasional lapse into complacency − a few software writers have convinced themselves that the last ROM issue was either MG or JS and that these contained specific routines in specific locations.

Whilst some were moderately excusable due to there being no safe and documented Trap or Vector to access them, there were others which seemed just plain silly − notably one product (no names, no pack drill) which assumed that an RTE instruction could always be found at a particular location.

Where the 'infringement' was excusable on the grounds of the routine being un-vectored, then we have tried to accommodate these products by making the ROM look as the software expected.

**Please note that using the two-screen facility causes a lot of software a lot of problems:** the very first thing to try if a package gives trouble is to run it in one-screen mode. Note also that the various memory-cut routines available often make assumptions about the ROM contents: see ASSEMBLER for details of the built-in memory cut routine, which can be used to replace most others.

We have tried many different software packages with Minerva: if you find you are having problems with a particular piece of software then we would be grateful if you could provide as much data as possible to help us work out where the problem arises. If it is a legitimate system call that is now failing due to an alteration in the way that Minerva works, we shall do our best to correct it.

## Technical Help

The best way of getting a quick fix to this sort of problem is to supply us with a copy of the offending software, with adequate instructions on how to reproduce the problem − a suitable BOOT file and a list of key-strokes is best. It's also useful to know what hardware you have. When we've fixed the problem, we'll return your medium, possibly with a patched version of the software on it. If convenient, it can be useful to try the same thing on an "official" ROM version − sometimes we find that an apparent Minerva bug occurs in JS as well!

## Wish-list

If you have any ideas on future improvements, the wish-lists are still open − we don't implement every idea (I HATE screen savers, so don't suggest them!), but everything is considered. We tend to prefer ideas which add to the extendibility of the system, rather than extend it in their own right: the SuperBASIC TRACE hook is a good example of this. Don't assume "somebody else must have thought of that idea", or try to work out in detail how we could do something − if you work out the "what" we'll attend to the "how" (assuming the idea gets past the "why?"!).

## Credits

Thanks as ever to Tony Tebby of QJUMP for his comments, Ian Stewart and Adrian Soundy of Liberation Software for their help with the workings of QLiberator and QLOAD and for producing a new version of QLiberator which supports the extra Minerva features; the members of QMAS, the local QUANTA sub-group for throwing their collective software libraries at each new version of Minerva for testing purposes and you for supporting our efforts to keep the QL alive! Thanks also to those who contributed to the wish-list via ATAVACHRON, other bulletin boards, letter, 'phone and carrier pigeon...

Minerva is now being distributed by Tony Firshman, who can be contacted at the following address − all technical queries that he can't answer will be passed on to the relevant QView bod.

TF Services
12 Bouverie Place
London
W2 1RB

Telephone: (071) 724 9053
Scrolling BBS: (071) 706 2379

# Fitting

You will need a cross-head screwdriver to undo the QL casing, and a chip extraction tool or small flat bladed screwdriver to remove the ROM chip set.

Open the QL

**1/** Remove screws from the QL casing − *except* the screws holding the microdrives. There should be eight − if you have ten then it's too late, you obviously have a deep-seated fear of reading instructions.

Find the old ROMs

**2/** Locate the two QL ROMs which are labelled IC 33 and 34 on the PCB, they should have printed on the top 'QL JS' or 'QL JM' and then either '0000' or '8000' which is the location in the memory map that the ROMs inhabit. If you have QJUMP's Internal Mouse Interface, IC34 was transferred to it − it's the smaller of the socketed chips.

**2a/** If you appear to have a piggy-backed set of ROMs with flying leads you have an EPROM'd QL which will need converting with the following set of instructions *before* Minerva will work correctly − or any of the later ROMS such as JM/JS/MG for that matter. Should you require the instructions on modifying your AH machine, here they are otherwise go onto Stage 3:

**WARNING- QVIEW cannot be held responsible for any damage you might do to your QL in the AH conversion. If you are at all doubtful then seek expert advice.**

**2b/** Remove and discard any flying wires from A14, IC34 and JU points.
**2c/** Remove all wire links in JU points 1 to 6 (to the right of IC34). Note that JU1 and JU6 may appear to be fitted with resistors with only a single black band. These are in fact zero resistance and were inserted because the automatic component insertion equipment used to populate the boards cannot handle bare wire links.
**2d/** Remove IC17 (74LS00) − on some PCBs this chip is socketed and on others it is soldered into the PCB.
**2e/** Remove IC33 and 34 (EPROMS)
**2f/** Fit wire links to JU2,JU3 and JU4

Take them out

**3/** Remove both of the ROMs gently to avoid bending the legs into wire sculptures. Store in a safe place as you might one day want a nostalgia trip....

Fit Minerva

**4/** The Minerva assembly plugs into the socket IC33 which is the one nearest the expansion port on the left. Make sure the new EPROM is fully home with gentle but firm pressure. The notch at one end of the chip should point towards the serial port sockets as the original ROMs did. *Do not* replace the ROM that came out of socket IC 34, this socket should be *empty,* even if it's now on your QIMI. If you plug Minerva in the wrong ROM socket don't worry too much, it works fine in either!

Enjoy

**5/** You can now replace the keyboard casing and power up your QL to enjoy the wonderful new Minerva logo....

**6/** Don't be unnerved if after 15 seconds the system appears to spring into life − Minerva has just got a little restless and decided to start without you.....

**7/** You can now press F1/F2/F3/F4 during the "tweed" pattern and Minerva will remember which screen option you required when the RAMTEST has been completed.

If at this point you see a block of large hexadecimal numbers then Minerva is signalling that there appears to be a RAM failure or fault of some kind. For details on what these numbers mean see under RAMTEST in the Assembler section. Other failures are rare, and likely to be a result of undue timidity in shoving the Minerva assembly into its socket − go on, give it some wellie. Don't be alarmed if you see a continuous white line in the tweed pattern which appears to move from time to time − this is perfectly normal and a result of the extra checks in the memory test routine.

## RTC version

Fitting is very similar to the original Minerva. The rechargeable battery can be placed anywhere convenient in the QL; we have allowed enough wire to put it in the spare corner of the case between the 64-way expansion connector and the back wall.

All Minerva RTCs are tested and pre-configured with the correct time. Mishaps do occur, however, and it is possible that the contents of the RAM will have become sufficiently scrambled to prevent the QL from starting. To get the system going, press and release the QL's reset button TWICE within one second or so: you should then see the usual "tweed" pattern.

In some rare circumstances, if corrupted Minerva ram is detected (vertical bars) the QL will fail to start, even after pressing left arrow, or double pressing reset. In these circumstances, short VBAT to ground (ie pin 1 to pin 2 on the 9 pin I²C external socket). Then re-configure the RAM.

## Configuring

Two versions of the configuration program are supplied − one is the SuperBASIC source file (MINICONFIG_BAS) which requires the I2C_IO_BIN extension loaded to operate correctly, the other is a QLiberated version (MINICONFIG_OBJ) which can be EXEC_W'd at any time without needing the SuperBASIC extension to be present.

The configuration program has the following options:

| | |
|---|---|
| ENTER | Review/alter the current settings. |
| T | Allows the date and time in the I²C RTC chip to be reset |
| L | Load a set of configuration details from a file |
| S | Write a set of configuration details to a file |
| R | Read the current configuration from the I²C RTC RAM |
| W | Write the settings to the I²C RTC RAM. |
| Q | QUIT the program. |

Pressing the **ENTER** key displays the current setting for the next configurable item and allows the user to alter it. To leave an item as displayed, simply press ENTER again to step to the next. The configurable items are as follows:

**NOTE all values prefixed by '$' are displayed/expected to be HEX!!**

RE-BOOT D1 Value
ROM Disable
NET Station
System De-Enhance (SX_TOE)
SuperBASIC De-Enhance (BV_TOE)
Type ahead string *

** Note that you cannot edit this. If you make a typing mistake you should re-write the complete string before writing it to the I²C RAM. Note if you have typed an F1 or F2 as the first part of your type ahead string, you may need to add a couple of spaces before the rest of the string that you would like typed into #0. If your configuration string is giving "BAD LINE" because it looks like some of the characters of your type-ahead string have been 'eaten' then add the spaces just before the offending statement*.

The various configuration items are described in more detail in the SuperBASIC and Assembler chapters.

Pressing **T** reads the current settings of the I²C clock and allows you to set it using the same parameters as for SDATE. Type the SDATE parameters separated by ENTER each time.

e.g:

```
1991<ENTER>3<ENTER>31<ENTER>12<ENTER>59<ENTER>0<ENTER>
```

On the last parameter, the I²C clock will be updated with the new date and the QL Clock will also be passed the same value via SDATE.

**L**OADING and **S**AVING Configuration files:
If you intend to change your configuration regularly, you might like to save the details under a number of different filenames so that it is a simple matter of LOADING the appropriate file and WRITING it to RAM to set up your new configuration. LOADING and SAVING prompt for a device and filename in one − e.g: flp1_MySetUp

**R**EADing the current settings allows you to check that the I²C RAM has been set according to your wishes. Use ENTER to review what the I²C RAM currently contains.

**W**RITE sends the current settings to the I²C RAM after prompting for confirmation. Once this is confirmed, the new settings will take effect from the next re-boot of the machine.

## Files

On the supplemental files medium you will find a number of items which don't really belong in the ROM, usually because they are of a more or less specialist nature:

MultiBASIC files − see SuperBASIC chapter

| | |
|---|---|
| MULTIB_EXE | MultiBASIC EXECable file |
| MULTIB_ASM | source file for the above |
| MULTIB_REXT | MultiBASIC command file |

Trace facility − see SuperBASIC chapter

| | |
|---|---|
| TRACE_TRACE_ASM | source files |
| TRACE_CHAN_ASM | for the trace facility |
| TRACE_LINK | linker command file |
| TRACE_MAC | macros |
| TRACE_KEYS | symbol definitions |
| TRACE_BOOT | boot file for... |
| TRACE_BIN | ...the resulting SuperBASIC extension! |

Utilities to improve compatibility − see Concepts and Incompatibilities chapters

| | |
|---|---|
| SVCHECK_BAS | checks for naughty software |
| BODGE_xxx | bodger programs for naughty software |
| QL_BIN | improved version of QLOAD/ QLRUN |
| TURBOFIX_xxx | To allow use of some Turbo extns in MultiBASIC |
| PUBLISH1_EXTS | Fixes problems with Pro Publisher |

Foreign keyboard drivers − see Concepts chapter

| | |
|---|---|
| GERMAN_ASM | source for German version of... |
| GERMAN_KBD_BIN | ...LRESPRable... |
| GERMAN_KBD_ROM | ...or ROMmable keyboard layouts |
| FRENCH_xxx | French... |
| MGD_xxx | ...Danish and Norwegian... |
| MGY_xxx | ...and Finnish versions of the above |
| KBD_ROM_BAS | Program to produce ROMabble code |
| AUTO/MAIN/SIN_INC | Include files for _ASM |
| COBBLER_BAS | Program to combine _BIN files |
| KBDTXT_BAS | Display of screen layout |

Test utility − see Concepts chapter

| | |
|---|---|
| RAMFAIL_BAS | shows faulty RAM chip on-screen − needs to be run on a good QL! |

I²C specific files

| | |
|---|---|
| I2C_IO_BIN | SuperBASIC extension to access RTC and other devices |
| MINICONFIG_BAS | Program to configure battery backed ram − SuperBASIC source... |
| MINICONFIG_OBJ | ...and compiled version |
| HANOI_xxx | Robot control programs |

Other utilities

| | |
|---|---|
| TWOSCREEN_DEMO_BAS | Two screen demo prog! |
| DEBOUNCE_xxx | To stop keybounce (Hermes is a better solution!) |
| QUIET_xxx | To shut down sound |

# Incompatibilities

## Problems

Remarkably few considering the range of changes we have introduced, but nothing is perfect and here are a few of our past problems some of which may not appear in the version of Minerva you have! Any revisions to the current state of compatibility will be found in the **updates_doc** file on the Minerva Technical Disk supplied with your ROM.

## Bodges!

Some programs and extensions have bugs in them, which previous versions of QDOS let slip through. Minerva being somewhat fussier can expose these bugs, and we have therefore provided a series of bodges to correct some of them. These are in the form of SuperBASIC programs, in files of the form 'bodge_program_name'. Inspection of the program will show the file it's expecting to operate on, and the expected length: if these are different from the version you have, the bodge program may not work. It's worth a try, though − just alter the name and length to suit, and try it! ON A BACKUP!

To bodge an offending program, load in the corresponding bodger program, put the offender in the drive expected (usually flp1_) or alter the bodger to suit you (e.g. to use a RAM disc), and run the bodger. Progress is reported, and the result written back to the source medium with '_bodged' appended. Try using this instead of the original, you should find the specified problems have been fixed. If not, please send us your version of the software and instructions on how to reproduce the problem − we'll return your medium with a new bodger and a suitably bodged version of the software you sent.

Note: the bodger programs are pretty slow in operation, don't worry about this, you'll only be doing it once. You can compile them if you like, unless your compiler won't cope or it's the compiler you want to bodge...

We at QView have a policy of informing the author or publisher of a program about any bugs we find. The majority of them will, in time, fix such bugs and provide updates for their customers. This may take some time, as most people like to collect at least a month's worth of bugs before doing any fixes, especially if the software's quite old. The advice here is be patient, polite, but persistent. Also, be very clear about which bug it is you need fixed − otherwise you may end up with a more recent version which is still no good to you.

To the bodges.....

## QLiberator runtimes

Some commercially available software, e.g. 4Matter and Locksmithe, has been compiled using a version of QLiberator which is now slightly out of date − the Runtime routines, which are often built into the program at compile time, make assumptions about the position of certain routines within the QL ROM. Most of the problem programs will also fail on an MG ROM issue as the way the ROM handled data in its registers was changed from the earlier issues. The long term solution is to obtain a more recent version of the software which should have been compiled with a more recent version of the Liberator Runtimes. As a short term measure, we have included a compiled program "QLibodge_obj" which will unlink any Runtimes which have been built in. When a QLIB program cannot see the Runtime code within itself, it looks for it elsewhere in the machine. You should RESPR and CALL, or LRESPR if you have TK2, your most recent Liberator Runtimes. If you do not have this file (why not?!) then use the Runtimes that are supplied to run "QLibodge_obj".

A typical "bodged" boot program might therefore become:

```
10 b=RESPR(10064):LBYTES 'flp1_QLIB_run',b:CALL b
15 b=RESPR(cde_size):LBYTES 'flp1_extensions',b:CALL b
20 EXEC 'flp1_a_program_obj_bodged'
```

If you try and execute a 'bodged' Liberator program without the Runtimes being present in the machine then you will get the error message:

*Runtimes Missing!*

In most of the programs we have tried this cures the problem with only a slight side-effect that each of the problem compiled programs is carrying a 'dead weight' of about 8K of code, hence it is really a short term measure.

## QJump Hotkey System 2

Hotkey system 2.06 and possibly some of the earlier versions give a problem with the extension HOT_LOAD, due to a missing '#' in the code, on Minerva ROMs the stack will be over-wound causing the machine to collapse either immediately or when something serious is attempted. The program "bodge_hotkey2" can be used to correct this to function correctly − QJUMP have been informed and it is unlikely that versions later than 2.06 will have problems.

## QLOAD QLRUN

In some cases we have found an intermittent problem with `QLOAD` when followed by `RUN` or `QLRUN`. The symptoms vary between hanging the machine or some spurious error message. In versions up to at least 1.6 of QLOAD/QSAVE, integer tokenisation is likely to give unpredictable results. We therefore recommend that you disable integer tokenisation with `POKE \\212,128` before using old versions of QLOAD/QSAVE.

QLOAD v1.7 is available from Liberation which fixes the intermittent problem, and handles integer tokenisation without difficulty or the need to turn it off in Minerva. For those commercially available programs such which use the QLOAD-only routine to speed program loading (such as Flashback

SE Report Generator) we have included on the utility disk a file QL_BIN which is the v1.7 QLOAD/QLRUN extensions supplied by Liberation Software. Note this does not contain QSAVE − if you regularly save and load large SuperBASIC programs we recommend you buy the full QLOAD/ QSAVE set from Liberation Software.

## Compilers

With the implementation of integer tokenisation in current releases of Minerva, there is a conflict with some versions of compilers produced by Digital Precision (Turbo and Supercharge) and Liberation Software (QLiberator). Unless you know that the version of the compiler you use has been modified to recognise this Minerva feature, you will need to disable integer tokenisation before loading and compiling your SuperBASIC program. The procedure is as follows:

```
POKE \\212,128      Turn off integer tokenisation
.....               Load SuperBASIC program
.....               Compile as per instructions
POKE \\212,0        Restore integer tokenisation
```

If you `LIBERATE` from a *_sav* file instead of producing a *_wrk* file by using the `LIBERATE` command, then you should ensure that integer tokenisation was turned off when the SuperBASIC program was first loaded and subsequently QSAVEd. The same caveat applies if you QLOAD a file prior to Turbo or SuperCharging it.

Liberation Software have now released v3.34 of their Q_Liberator compiler which is able to handle the integer tokenised format of SuperBASIC programs in Minerva. This is also happy in the two screen environment, and supports WHEN ERRor. This means that with this version it is no longer necessary to turn off tokenisation before loading SuperBASIC programs, To date we have no news of a Turbo/Supercharge upgrade from Digital Precision which is integer tokenisation compatible.

Integer tokenisation also has an effect when RENUMbering lines of SuperBASIC with line numbers less than 127. See the `RENUM` section in the SuperBASIC chapter for further information.

## Turbo Toolkit and Illegal Extension Names

Some versions of the Turbo Toolkit (also to be found in some commercial software packages called EXTRAS or XTRAS) contain the illegal extension names FUNCTION and PROCEDURE. Minerva will reject such illegal names (and other extension names with silly characters) and return from BP.INIT ignoring any subsequent procedures or functions. Digital Precision have produced a v3.30 Runtime Turbo Toolkit which no longer contains these illegal extension names. We have included the Runtime version on the utility disk as NEW_TURBO_EXTRAS. For the full toolkit for use with the Turbo compiler please contact Digital Precision.

TURBOFIX_EXT is an extension to allow certain TURBO keywords to be used in MultiBASIC. (Rich Mellor public domain utilities). Put the following in your boot file *after* loading Turbo extensions:

```
a=RESPR(1256):LBYTES flp1_TurboFix_ext,a:CALL a
```

## Pro Publisher

There are files to solve problems with Pro-Publisher. PUBLISH1_EXTS must be RESPRed with a size of 62800 despite its much smaller size...

## Solution and Conqueror FORMAT

Minerva now has an extended check before a FORMAT call, to avoid allowing a FORMAT if any channels are still open to that medium.

The MS.DOS emulators Solution and Conqueror from Digital Precision leave a '*d2d' direct sector access channel open all the time so under Minerva a FORMAT within Solution or Conqueror will fail. To deal with this we have implemented sx_toe to turn off such enhancements. To allow these emulators to FORMAT disks you will need to set bit 7 of sx_toe with the following POKE:

```
POKE !124!49,128
```

## SDUMP

SDUMP is a bit sneaky in that it parasites itself on a job to do its I/O operations for it. Under normal circumstances SuperBASIC will quite happily do this. However in order to create multiple SuperBASICs and the concept of an interpreter class job we used an available bit in the Job Header − the JB_RELA6 ($16) offset − with bit 6 set if we're one of these interpreters.

Tony Tebby's code should be testing the MSB but tests with BNE instead of BMI which tests just the top bit for you. You can get an amusing effect with JS etc. − if you start an SDUMP, then hit CTRL-SPACE the dump will stop. EXEC'ing something like QUILL will then restart the dump!

As to a cure... well, the best thing to do is have a job running which doesn't disturb the screen and has the JB_RELA6 byte set in the way SDUMP expects to test. Just such a job is, by a happy coincidence, FSERVE! So have FSERVE running before you issue the SDUMP command. We don't think there is any way round this in current Minerva, we've passed this information on to Tony and he will doubtless arrange for it to be cured for the next issue of the Trump Card TK ROM.

## Minerva RTC and Gold Card

Any Minerva RTC with a yellow sticker on the PAL and v1.92 or above is compatible with Gold Card.

## Hatemail!

We're told that people found the following file on our early documentation files most amusing so we've incorporated it into the printed version!

People we really HATE...

...those who assume there's an RTE at address $5E in the ROM, because they're too lazy to write the code to fill in their trap redirection table
...those who put filenames in "spare slots" in the system variables,because they're too lazy to work out a legal way of communicating between passes of their "compiler"
...those who assume the system variables live at address $28000
...those who assume the screen is at address $20000
...those who write directly to the screen, without using the SD.EXTOP trap
...those who open files for writing when they're only going to read them
...those who use the wrong trap to allocate memory in the system heap − the correct one is MT.ALCHP, not MT.ALLOC

People we slightly hate...

...those who search the ROM for a known pattern, but don't start at the beginning "to save time"
...those who call routines which they "know" start just after (or before) a legally-vectored one
...those who set the MODE without reading it first to see if it's already the one they want
...those who think it's worth using priorities above 127 to get a few percent extra speed (note our change in this area!)

# Concepts

Startup

Some (!) changes have been made to the start-up routines. The RAM test is faster, which will please Trump card owners. In order to make the RAMTEST more reliable, the starting point for the “tweed” pattern of random bits is started from one of 4096 possible points. Thus a few consecutive presses of the RESET button should make sure that your RAM is safe from most errors including the elusive refresh problems which may only show as sporadic locking up of the machine for no readily apparent reason.

RAM test failure

Should the RAM fail the test, you will get three lines of information on the screen: the value which was written, the value read back and the memory location at which the failure occurred. There will be a pause of about ten seconds before Minerva restarts stepping the memory size down to just below the failed area. In an extreme case of a failure of an internal RAM chip you may find yourself with a 48K QL − just be thankful you started with a ZX81 once! See the RAMFAIL_BAS utility on the supplied Utility disk.

If all your memory passes its physical, you then have the option of pressing F3 or F4, which goes through some of the start-up code again in order to enable the second screen − we couldn’t figure out a way of moving the system variables without bringing the system down around our ears, and leaving them permanently at the “second screen” location would confuse the (badly-behaved) software that assumes the old location. F1 and F2 have the original effect of putting you in monitor or TV mode − in combination with SHIFT the memory is cut to 128K, with CTRL the ROM scanning is omitted for really badly-behaved software.

After F3 or F4, pressing the screen switch key CTRL-TAB should now give you a blank screen instead of the pretty coloured dots and things that appear when you screen switch on a single screen Minerva.

Auto-start

If you don’t press F1 or F2 within fifteen seconds of the boot screen appearing, the system will start anyway, pretending that you’ve just pressed the F2 key. This will be of use to those who leave systems running while they’re out − they’ll re-boot automatically after a power cut. If you wanted F1 as a start-up then just add the following magic mode command to your BOOT program − MODE 4,0. *Note that this just resets the hardware* − if you want the windows changed to the appropriate size you’ll have to do that yourself in the boot file. If you need to perform a software reset, you can now do so: see the Assembler section of this guide for details. You can also do a reset from the keyboard, with CTRL-ALT-SHIFT-TAB.

RTC startup

If you have Minerva RTC the clock will always be copied into the QL’s 32-bit seconds counter whenever the QL is re-started. You can prevent the other configuration information (see the SuperBASIC chapter) from being used by pressing the left-arrow key after reset but before the F1/F2 etc. message appears − useful if you want to check what ROMs you have plugged in without changing the configuration, for instance.

## Keyboard changes

A number of changes have been made to the keyboard, to improve usability and gain access to some of the new facilities. The following list of keys did nothing (useful) in previous versions: while the functions they now perform are (where appropriate) retained on their original keys, these ones are hard-wired into the system and can't be modified by POKEs. This is especially useful for the new "next job" key, as CTRL-C is forever being zapped by unfriendly software, and never twice the same key either!

| Keystroke | Function | Old keystroke |
|---|---|---|
| CTRL-ALT-SPACE | BREAK MultiBASICs | (none) |
| CTRL-TAB | swap displayed screen | (none) |
| CTRL-ALT-TAB | screen freeze | CTRL-F5 |
| CTRL-ALT-SHIFT-TAB | Keyboard RESET | (none) |
| CTRL-ENTER | compose character | (none) |
| CTRL-ALT-ENTER | keyboard queue | CTRL-C |
| SHIFT-CTRL-ENTER | CAPSLOCK | CAPSLOCK |
| ALT-CTRL-SHIFT-ENTER | Call User routine | (none) |

## Compose character

The only really non-obvious one of these is compose character, CTRL-ENTER. This allows you to type in that tricky foreign character you know is in there somewhere, but is it on CTRL-= or CTRL-SHIFT-1?! Now all you need do is type CTRL-ENTER, A, : for a-umlaut (an a with two dots, OK?), and so on. Where an upper-case version exists, shifting either of the two characters gives the upper-case result (or having caps lock on, of course). We've tried to keep the combinations pretty obvious: \ and / combine with letters to give grave and acute accents, : for umlaut, and ^ (or 6) for circumflex. We've avoided the quote key, as it's not obvious whether it adds an accent (´) or umlaut (¨). Note that symbols (^, : etc.) are added correctly whether or not you press the SHIFT with them: you get a umlaut from CTRL-ENTER, A, ; as well.

The compose table is currently as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| á | a/ | ô | o^ | ø | o! | ¡ | !! |
| à | a\ | ú | u/ | ü | u: | ¿ | ?? |
| â | a^ | ù | u\ | ç | c, | § | pp |
| ë | e: | û | u^ | ñ | n~ | ¤ | ox |
| è | e\ | ß | ss | æ | ae | « | << |
| ê | e^ | ¢ | c! | œ | oe | » | >> |
| é | e/ | ¥ | y- | α | aa | ° | oo |
| ï | i: | ` | \\ | δ | dd | ÷ | :- |
| í | i/ | ä | a: | θ | tt | ← | the |
| ì | i\ | ã | a~ | λ | ll | → | appropriate |
| î | i^ | å | ao | μ | mm | | cursor |
| ó | o/ | ö | o: | π | pi | ↓ | key |
| ò | o\ | õ | o~ | φ | ph | | |

## Enhanced movement

**IO.EDLIN** which is called by EDIT, AUTO and INPUT can now accept enhanced movement keys which are as follows:

| | |
|---|---|
| ALT ←/→ | move to start/end of current line |
| TAB | move along to 8th character from start of buffer |
| SHIFT-TAB | moves BACK in same steps as above |
| CTRL-ALT ← | deletes to start of current visible line |
| CTRL-ALT → | delete from current character to total end of line |
| ESCape | behaves like CTRL/SPACE (Break) |
| SHIFT-ENTER | behaves pretty much like ENTER |
| SHIFT-SPACE | behaves pretty much like SPACE |

These are available to all software that utilises the "IO.EDLIN" trap to fetch lines from a "con" channel. E.g. "EDIT", "INPUT" and a large percentage of other software.

Cursor movement:

The arrow keys, with no shift keys held, normally move the cursor in the direction indicated. When left or right reach the ends of the data of a line, they stop there. If a cursor up keypress would move onto the first row of the line, but would be to the left of the start of the line data, it will move to the start of the line. Similarly, if a cursor down would move onto the last row, but beyond the end of the data, it is just moved to the end. Should cursor up or down be pressed when the cursor is already on the top or bottom row respectively, they terminate the input (see below).

The "TABULATE" key moves forward to the next multiple of eight characters, counting from the start of the line, stopping should it reach the end of the line. With "SHIFT", it will move back to the same points.

The left or right cursor key, with just "SHIFT" held, will move by a word, i.e to the next "word start" in the appropriate direction. The start and the end of the line data are considered "word start"s, along with any position within the line data where the character before the cursor would a space and the character under it would be other than a space.

The left or right cursor, with just "ALT" held, will move to the end of the line in the appropriate direction.

Character deletion:

All the above combinations of the left and right cursor keys, with or without "SHIFT" and "ALT", when combined with "CTRL", will delete all the characters that would have been moved over.

Note: unlike the previous "IO.EDLIN", where deletions result in the line data occupying less of the screen, the new version wipes such space to the current "PAPER" colour. The old "IO.EDLIN" wrote spaces, which would come out as the "STRIP" colour, which could be confusing.

Character insertion:

All characters in the range CHR$(32) to CHR$(191) are inserted in the buffer and displayed appropriately. For characters outside that range, pressing the space bar with "SHIFT" held is treated exactly as if the "SHIFT" key was not held. I.e. CHR$(252) is converted to CHR$(32) immediately.

Line termination:

If the line buffer is full, or the timeout expires, an immediate return from "IO.EDLIN" occurs, leaving the cursor on, and giving the appropriate error code. The buffer overflow is used by SuperBASIC to prompt it to increase the buffer size and re-enter the edit. This is fairly transparent to the user.

If up or down cursor is used to leave the line (as described above), or the "ENTER" key (with or without "SHIFT") or "ESC" key is pressed, the cursor is moved to the end of the line (displaying whatever is needed), the terminating keypress character is put in the buffer after the last of the data, the cursor is turned off and a normal return is made.

Pressing "ENTER" with "SHIFT" held is treated exactly as if the "SHIFT" key was not held. I.e. CHR$(254) is converted to CHR$(10) immediately.

Display handling:

There was always a problem if the cursor position was not at a "standard" character position or the first character of the line being edited did not come at the left edge of the screen.

In the latter case, one has a problem if the first row of the line was scrolled off the top of the window. Previously, it was impossible to get back to there, but the new IO.EDLIN allows this. However, it is not possible to redisplay whatever might have been before the start of the line (e.g. a prompt). The is no solution to this problem, so the new IO.EDLIN just leaves "PAPER" colour there.

The nastier problem occurs when the cursor is not at a "standard" character position ( i.e. one reachable by an AT command). Some software "solves" this by forcing the cursor to such a position first. If used without such forcing, the old IO.EDLIN would work fine until the cursor was moved forward off the initial row, and then get rather messy if it was moved back. The new code sticks rigidly to the initial cursor position, only every displaying characters at exact offsets relative to that position. This can result in a reduction by one of the effective width and/or height of the window, but is altogether more tidy overall.

Some of the display strategy is a little "quirky", but overall it is reasonably friendly.

## Dual screen

Whilst there is a perfectly good QDOS Trap MT.INF to find the location of the system variables and despite the QL Technical Guide stating that there is no reason why the System Variables should always be in the same place, many software writers seem to have decided that they always know they'll be at $28000. We have provided a small SuperBASIC program called "svcheck_bas" which you can use on any of your own commercial programs to discover whether they are usable in a two screen environment. Note that we've extended VER$ to provide VER$(-2), which tells SuperBASIC programs where the System Variables are. There is now NO EXCUSE!

It may be possible to patch a version of these offending pieces of software to use the position of the System Variables when the two screens have been enabled. This is currently $30000 (or higher if you use some of the strange options on CALL 390!). *But* we might change it.

CTRL-C (or the new equivalent CTRL-ALT-ENTER) now has the additional function of switching screens if the next active cursor happens to be on the other screen. Note the difference between this and CTRL-TAB, which allows you to inspect the other screen, without moving from the current job.

## Screen Tweaks

Apart from implementation of the second screen, a number of improvements have been made to screen handling. 8-pixel-wide characters now work in all character sizes. If you use the Super Toolkit II CHAR_INC routines, then characters will print out on a STRIP of the size specified, but won't fall out of the window as they can on previous ROMs. This was particularly embarrassing when the STRIP fell out into the system variables! The character set has been extended to include a complete (if rather scattered) Greek alphabet and some other "useful" bits and pieces. It's a moot point whether this is a useful way of occupying 1k of valuable ROM space, so don't rely on them. We'd welcome comments on what people want in an extended character set: we now have ways of installing new character sets so they can be used by *all* jobs − see **sx_f0**.

## Screen Graphics

Graphics have been speeded up, and their robustness improved: ELLIPSE and ARC in particular benefit − you no longer get gaps in wavy lines drawn with ARCs, and narrow ellipses don't go bananas so readily. Even POINT has been speeded up! The net result is that graphics are now at least 98% of the speed of the LIGHTNING graphics, without taking up 4K of your memory. (Figures are based on DP's own graphics benchmark, running LIGHTNING 1.13 from RAM − a ROM-based version would be faster).

## MDV date-stamping

On a Miracle machine without Toolkit II the system will now date-stamp both the creation and update dates of files on the microdrive. Whilst we would have liked to implement these concepts on floppy disk drivers, these are not controllable by the operating system as the external device driver over-rides the internal routines.

## Network Broadcast

For machines without Toolkit II (still not convinced you to buy one from QJump yet ?!) the broadcasting to all stations listening has been made more reliable than previous issues but is not yet absolutely 100% perfect in all situations.

## Scheduler

Break (CTRL-SPACE) handling has been moved here to avoid having it inside interrupt service code. Also, CTRL-ALT-SPACE breaks all interpreters other than job 0. The usage of job priorities has been enhanced somewhat. They now behave as:

| | |
|---|---|
| - 128..-1 | "background" tasks (see below) |
| 0 | job is inactive |
| 1..127 | major active jobs, as before |

Background tasks are split into eight levels, according to their top nibble, within each of which the low nibble now gives the priority increment.

Background tasks of a given level are given time only if no major job and no tasks at a higher level want time. Note that their negative priorities may be reported by some utilities as large positive numbers, so a task with a priority of -1 may be reported as 255.

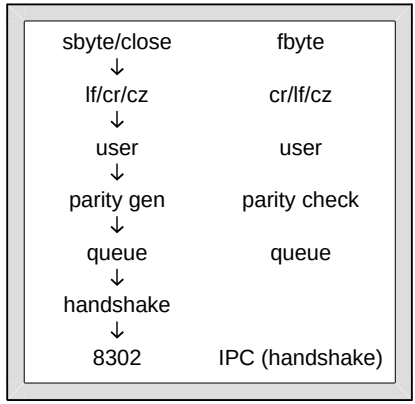
## Serial driver

This has all been shuffled about quite a bit, in an attempt to get toward properly functioning serial channels. The current version still suffers from some problems.

The main problem is that handshaking can only be actioned at the stage of actually sending bytes to the 8302. Swapping between 'SERi' and 'SERh' opens will not get this right. However, this seems to be the least of evils, as one will usually be changing the hardware plugged into the port in these cases. All other handling is done earlier, including making the CLOSE do generation of CTRL-Z if required.

The serial queues now contain only raw data to/from the serial ports. The action of “SERc” is now treated as an exchange of CR/LF values. The I/O translation routines now occur between parity/handshake and protocol (CR↔LF and CTRL-Z) handling.

The only user translation that is difficult/impossible at the moment is the one-to-many conversion of data coming from the serial queue. The current sequence of operation is as shown in the funky grey box →




The channel structure is not discarded when closing a serial channel, as there may be further data pending in the transmit queue. This deserves more thought! The above sequence is not wildly satisfactory, as it can be held up indefinitely by a “SERh” channel with the handshake holding off output permanently. The original technique relied on outputting the CTRL-Z when the 8302 had emptied the queue, but this was prone to the syndrome of changing protocols before the code had actioned the protocol. For instance: sending a series of small CTRL-Z files, then changing to “SERr” would not send any CTRL-Z’s! An alternative scheme to avoid the handshake problems suffers from the same flaw as the above. This would be to use a special byte in the output queue as an “escape” character. A preferred value would be a zero byte. The output would then double up single nulls when they were real, say, and restore them in the IP routine. Other byte values after a null would be used to pass open/close parameters down.

## Foreign Keyboard Drivers

Provision has been made for foreign keyboard machines by allowing an appropriate keyboard driver to be RESPR’d into the machine at boot-up.

Once this has been done, the error messages and keyboard mapping will have been altered appropriate to the language of the driver installed. There are currently versions available for the German (MGG), French (MGF), Finnish (MGY) and Norwegian (MGD) ROMS. For those who have access to an EPROM programmer and who do not use the ROM port at present, there are *_ROM* files for each keyboard driver which can be blown into an EPROM so that your keyboard driver is available immediately on power up.

With a product like RPM from Liberation Software or Thing and EPROM Manager from Jochen Merz, you can incorporate the *_BIN* keyboard files into whatever EPROMs you have already produced.

For those without access to EPROM programmers, we would be prepared to supply the keyboard driver blown into an EPROM cartridge for a nominal fee − please contact us at the usual address if you require this.

## ABC Keyboard Interface Driver

For those QL users who have the ABC Keyboard Interface and would like to use Minerva on a QL fitted with this interface, we have provided a Minerva compatible driver on the disk called ABC_KBD_BIN.

This file should be the first thing loaded by your boot file if you want to use the keyboard. Unfortunately you cannot press F1/F2/F3/F4 until the keyboard driver has been installed so you will have to rely on the ten second timeout starting up your boot file unless you rig up a piece of wire with a switch to the old QL keyboard connector to allow you to press F1 or F2 from the QL keyboard. Note that both drivers can run in parallel so if your QL still has its keyboard attached you can use either the ABC or QL keyboards.

If you do not have Toolkit II with the LRESPR extension then you need the following sequence to load the driver:

```
10 base=RESPR(750):LBYTES flp1_abc_kbd_bin,base:CALL base
```

If you do not use the ROM port for any toolkit ROMs etc, you might consider blowing the ABC_KBD_ROM code into an EPROM so that it is initialised immediately on power up. If you do not have access to an EPROM programmer to do this but would like to install the keyboard driver in this way, please give us a call or drop us a line and we should be able to do it for you for a nominal charge.

The source file ABC_KBD_ASM is also provided for those who might find it of interest.

## Pipes

The full syntax for the pipe device is now:

```
PIPE[<IDin>][X|P|T]<IDout>[_[<length>]][K]
```

"IDin", "IDout" and "length" are all decimal values in the range -32768 to 32767 and if omitted, they default to zero, with one exception. If one of "X", "P" or "T" is given, and "IDin" is present, and not zero, "IDout" will default to (or a negative value be forced to) the same value as "IDin".

If "K" is given, the "length" is multiplied by 1024 to give the actual length of pipe to be created. (Note: the length of a pipe is the exact number of characters that it can have written into it before it will be full, and need someone to start reading them out.)

Old style pipes:

If none of "X", "P" or "T" is given, and "IDin" is omited, or zero, then old style pipes are used.

Firstly a channel must be opened to write to the pipe, specifying a length greater than zero for the length of the queue to be created. A second channel must then be opened to read from the pipe, by giving a length less than or equal to zero, or omitted. The QDOS channel number of the first channel must be passed in D3.w when the second channel is opened.

An aside: on a totally bare machine, it was just possible to connect up pipes, provided you didn't mind losing one of #0, #1 or #2. E.g.

```
OPEN#2;'pipe_100':OPEN_NEW#6;'pipe'
```

would work, as the open code IO.NEW is 2, and QDOS channel number 2 just happened to be associated with SuperBASIC's #2. Very nasty stuff!

The facility to specify long pipes via using "K" is available even on the old style pipes. Also, the connection is much more thoroughly checked. Only one input channel at a time is permitted and connecting input pipes to one another is rejected (both such lunatic things could be done previously, resulting, eventually, in crashes).

New style pipes:

These come in when either "IDin" or "IDout" is non-zero.

The first channel opened to an ID must specify a length greater than zero for the length of the queue (or queues) to be created. On any subsequent opens using this particular ID, the length is totally ignored.

There is no limit to the number of channels inputting from and/or outputing to the same ID pipe.

If "IDout" is zero (or omitted), and one of "X", "P" or "T" is given, the channel will be read-only.

If "IDin" is omitted, the channel will be write-only.

If both "IDin" and "IDout" are non-zero (or "IDout" was made to duplicate "IDin", as described above), the channel will provide data from "IDin" and data sent to it will go to "IDout". The can be the same ID, in which case a single channel can be used as a "first in, first out" circular queue.

Normally, when the last channel capable of writing to an ID has been closed, the queue is marked as at "end-of-file" and becomes anonymous. The ID is then available for re-use. When there are no channels at all connected to an anonymous queue, any data still in it is lost and its memory is returned to the system.

The "X" is just to separate "IDin" and "IDout", but a "P" (permanent) will ensure that the pipe (or both pipes) are preserved even if all channel connections are closed. A "T" (temporary) will revert from this state.

Should one wish to open a channel to two distinct IDs, and want their lengths to be different, one has to has to open a dummy channel to one of IDs first, in order to define its length. The required channel can then be opened to create the queue for the second ID, and the dummy channel is then closed. Alternatively, only a single channel need be used, if the "P" and "T" flags are used. E.g.

```
OPEN#3;'pipe1p_100':OPEN#3;'pipe1t2_20
```

As a recommendation, a job which is creating pipes should incorporate its QDOS job number in the IDs that it uses. The suggested system is to use the hundred IDs starting at the job's own job number times a hundred. E.g. `'pipe'&j&'12'`, where `j` is calculated by

```
j=VER$(-1):j=j-INT(j/65536)*65536
j=(PEEK_L(!!100)-PEEK_L(!!104))DIV 4)
```

or any other convenient way.

Throwaway pipe:

"`pipet`" is a device which will permanently give "end-of-file" on reading and will discard any output to it. Any number of channels may be open to this simultaneouly.

Program pipe:

"`pipep`" is device that is read-only, and contains a copy of MultiBASIC, suitable for "`EXEC[_W]`" (or anything else that likes a serial stream input with a proper header showing the type executable, etc). Any number of channels may be open to this simultaneously and they will each receive the full data.

Some examples:

I quite like:

```
OPEN#3;'pipe2x1_100':OPEN#4;'pipe1x2':EXEC'pipep',#4
```

You can drive your own "floating" copy of SuperBASIC by PRINTing commands to #3 and seeing what comes back by INPUTing from the same #3.

A couple more examples:

```
PIPE1X_10       an input only pipe
PIPEX1_10       an output only pipe
```

A limited semaphore:

```
OPEN#3;'pipep1_10':PRINT#3;'xy':CLOSE#3
```

This has a initial value of three and a limit of 10. It can be useful for resource management, etc. Any program can come along and obtain a unit with:

```
OPEN#3;'pipe1':a$=INKEY$(#3):CLOSE#3
```

To release a unit:

```
OPEN#3;'pipe1':PRINT#3:CLOSE#3
```

To finally discard the semaphore:

```
OPEN#3;'pipe1t':CLOSE#3
```

## Minerva RTC

Minerva RTC adds a Philips PCF8583 real-time clock and RAM chip to the QL, and uses some of its contents to configure the QL at reset. The interface to this chip is the Philips I²C serial bus, driven by software at near its maximum speed of 100kbits/second. Although the original specification of this bus allows for multiple masters, the Minerva RTC implementation is restricted to a single bus master, the QL itself.

The clock and RAM can be accessed from machine-code or SuperBASIC. A vector to perform I²C bus transactions is provided in the Minerva ROM, and a SuperBASIC function that uses this is provided on the utility disc.

HANOI_MIN_BAS is a program to drive I²C parallel and analogue interfaces, to work the Fischer-Technik robot. Technical details and circuit diagrams are available for £3 (UK), £4 (EC), £3.50 (Europe) and £5 (ROW).

Pinouts for the external 9 pin 'D' connector are:

| | | |
|---|---|---|
| 1 | VBAT | |
| 2 | GND | |
| 3 | SDA | |
| 6 | VCC | |
| 7 | SCL | |
| 8 | INT | (external interrupt) |

4/5/9 are reserved for future use.

Several I²C add-ons are available; these include 16-line TTL/LED drivers, an 8 in/2 out analogue interface, a 4 output relay driver for mains or DC, and a high-current/high-voltage DC driver (e.g. for small electric motors). Several such units may be connected at once. Contact TF Services for more details.

## Hermes Support

The old 8049 code never attempted to interrupt the main processor (68K), which it was probably intended to do. This interrupt was not strictly needed, as 19200 baud receive was not supported anyway. Bearing this in mind, the code for servicing the "IPC interrupt" was removed from Minerva.

With Hermes, 19200 baud receive is now functioning, which means that (with about 1/2 stop bit) up to 41 characters may arrive in 1/50th of a second. Hermes can only buffer 31 characters, and indeed, it may only send 25 at a time to the main CPU, to maintain compatibility with old QL ROMs. In order to try to get the main CPU to keep taking characters, Hermes attempts to interrupt every eighth character it receives. This figure should usually keep the flow of data going without the handshake line being used.

From version 1.95, the IPC interrupt code has been re-instated to keep awake to Hermes.

Another facility of Hermes is that it can operate the serial inputs at differing baud rates, independent of the output baud rate. Version 1.95 has a similar facility built into BAUD (and MT.BAUD), allowing the output baud rates to be independent. At the extreme, the standard BAUD (or MT.BAUD) calls can now be totally ignored by the system, maintaining all four serial lines (in/out ser1/2) at four completely independent baud rates.

## Atari QL emulator and Thor machines

At the time of going to press, no specific work has been done to produce a version of Minerva to run on the Atari QL Emulator from the Futura Datasenter supplied by Jochen Merz. If there is sufficient demand we will try and find some odd moments to see how much work is required and whether there is sufficient interest to warrant it. The same applies to variations of the Thor family.

## Minerva versions

**The specifications in this manual apply to Minerva versions from 1.96 onwards**. If you have an earlier version then some of the features described may not be implemented on your version; if you feel you need them, please contact us for upgrade information.

Later versions of Minerva may have extra enhancements which we will describe in a printed supplement if the changes are major, or an updates_doc file on the utility disk in the case of minor tweaks.

ABS

The ABS function now takes a list of numeric parameters, and returns the square-root of the sum of the squares of its parameters. The observant among you will spot that this leaves its original function unchanged, though of course the one parameter call is *not* done by squaring and square-rooting the parameter!

ATAN

This can now take two parameters, ATAN(x,y) giving the angle from the origin to the point (x,y)... much the same result as ATAN(y/x) but not overflowing if x happens to be zero, and getting the quadrant right.

AUTO
EDIT
INPUT

IO.EDLIN which is called by EDIT, AUTO and INPUT can now accept enhanced movement keys which are as follows:

| | |
|---|---|
| ALT ←/→ | move to start/end of current line |
| TAB | move along to 8th character from start of buffer |
| SHIFT-TAB | moves BACK in same steps as above |
| CTRL-ALT ← | deletes to start of current visible line |
| CTRL-ALT → | delete from current character to total end of line |
| ESCape | behaves like CTRL/SPACE (Break) |
| SHIFT-ENTER | behaves pretty much like ENTER |
| SHIFT-SPACE | behaves pretty much like SPACE |

BAUD

As well as the standard baud rates (75, 600, 1200, 2400, 4800, 9600 and 19200), the special code may be given as a parameter to control independent output baud rates. The additional values accepted are small negative numbers, in the range -128 to -1. As a bit pattern, it is:

| Bit | Add | Function |
|---|---|---|
| 0..2 | 0..7 | baud rate − 0=19200, 1=9600,...7=75 baud |
| 3 | − | Always 0 |
| 4 | 16 | Set to force this port to ignore ordinary BAUD calls |
| 5 | − | Always 0 |
| 6 | 64 | Set to signal SER2. else SER1 |
| 7..15 | -128 | Signals new format baud control word |

Bit 1 of SX_TOE may be set to disallow these extended commands. If you don't have Hermes, then you should only use this command if you only want serial *output*, as it totally confuses input. See your Hermes documentation for how to split the input baud rates.

BLOCK

The screen driver entry (SD.FILL) for this command has been enhanced to accept any 16-bit signed integer values for width, height, x and y. Normal usage is unaffected, but one can use it very similarly to the graphics routines, in that it will now just fill in any part of the area that lies within the screen.
e.g. `BLOCK 200,20,-195,-10,7` will behave as `BLOCK 10,5,0,0,7`.

The reason for this enhancement is twofold. We found it quite irritating to

have to be so finicky (some of you will know what I mean). Secondly, we wanted something that would accept "BLOCK 2,2,-2,0,7", and various others, that JS fails to error trap correctly. This was causing trouble in some software that used such invalid BLOCK commands, and got away with it. (That is they did, before Minerva started to check values correctly.)

Be careful if you try out the duff BLOCK commands on non-Minerva ROMs; the example above actually draws itself on the righthand edge of a full size screen, but we wouldn't guarantee that other parameters won't cause a crash!

## DATE SDATE

The date procedures now accept a wider range of parameters. DATE accepts the six-parameter format formerly used by SDATE, allowing you to convert this form for use with DATE$ and DAY$ and SDATE now accepts a single parameter in addition to the original six parameter syntax. So:

```
PRINT DATE$(DATE(1962,3,21,9,0,0))
```

will print

```
1962 Mar 21 09:00:00
```

and more usefully

```
PRINT DAY$(DATE(1962,3,21,9,0,0))
```

will print

```
Wed
```

A small program to copy the clock from one QL to another across the network is now:

```
100 remf$="n1_ram1_find_the_time"
110 e=FOP_NEW(remf$)
120 IF e>0 THEN CLOSE #e:SDATE FUPDT(\\remf$)):DELETE
    remf$
```

Note that you do need the wonderful Super Toolkit II on both machines to use this.

## EXEC EXEC_W

These are extended to allow channel numbers and an optional string to be set up for the job. This is a strictly checked subset of the TK2 standard.

```
EXEC[_W]<filename>[,#<chan_no>]...[;<string>]
```

All the channels must already be open. They are passed to the job, along with the specified string, in the standard QDOS manner. I.e. the job will find the count (word) of the number of channels passed on top of its stack. The channel identifiers (longwords) follow this. Next if the string length (word) followed by the text of the string (bytes). If the length is odd, there will be one final padding character. Any space required for these optional parameters is added to the reqested dataspace for the job, to ensure that it does not confuse the job.

## FILL

This is a bug-fix. Previously, if a program didn't do a FILL #n,0 before a CLOSE #n, the fill buffer (used to keep track of which parts of the screen are to be filled) was left behind, resulting in 1k of memory being lost to the system. Now the fill buffer (if any) is automatically thrown away when a window is closed. (Sad, but with QJump's Pointer Interface loaded, the bug is put back...) Also the rules for FILL are slightly different and correspond more closely to the original manual.

## MODE

MODE now allows you to use both screens. The original one-parameter call is exactly as before. The new form is:

```
MODE screen_mode,display_type
```

Screen mode accepts the normal 4, 8, etc, but with a few additions (see below). Display_type is simple, 0 for monitor, 1 for 625-line TV, and 2 for 525-line TV, or (usually) -1 to leave the display type alone. Very little software uses the display_type record, as old versions of SuperBASIC smashed it!

Screen_mode is *very* complex...
The two screens are known as Screen0 and Screen1. Each screen may be in 4-colour mode or 8-colour mode, or blank. Each job now has a "default" screen on which any new windows will be opened − it can change this default to have windows open in both screens. The screen which is not the default for the current job (the one calling the new MODE) is the "other" screen. Note that is is not necessarily the case that the default screen for a job is the same one the user is looking at (the "displayed" screen).

Oh, and the user can't necessarily *see* the displayed screen, it might be blank. Your head hurt yet?

Toggling:

```
MODE 64+n,-1
```

toggles various attributes of the display, where:

| **n** | **Toggles...** |
|---|---|
| 1 | other screen from visible to blank |
| 2 | default screen from visible to blank |
| 4 | other screen from 4-colour to 8-colour |
| 8 | default screen from 4-colour to 8-colour |
| 16 | displayed screen from Screen0 to Screen1 |
| 32 | default screen from Screen0 to Screen1 |

You can add together the various values for n to combine effects, so n=12=8+4 will toggle the colour modes of both screens. Note: a change to the default screen takes place before any of the others. Also, none of these calls do the "forced re-draw" imposed by the one-parameter version of the MODE call.

Setting:

```
MODE -16384+128+k*n+c,-1
```

sets or resets display attributes. The values of n are as above.

| **k** | **Sets...** |
|---|---|
| 0 ... | to the “from” column above |
| 1 ... | to the “to” column above |
| 257 | toggles, as above |

Values of k can’t be combined. The c portion controls which screen is force re-drawn:

| **c** | **Re-draws...** |
|---|---|
| -16384 | other screen |
| 32768 | default screen |

You can add the c values to re-draw both screens. So...

| | |
|---|---|
| `MODE 80,-1` | toggles the “displayed” screen (80=64+16) |
| `MODE 96,-1` | makes subsequent OPENs happen on what was the “other” screen (96=64+32) |
| `MODE 112,-1` | does both the above simultaneously (112=64+32+16) |
| `MODE 16560,-1` | sets default to Screen1, displays it in 4-colour mode, and force re-draws all windows in it (16560=16384+128+1*(32+16)+32768) |

## OPEN
## OPEN_IN
## OPEN_NEW

Minerva will accept a third parameter on these commands. Should it be given, it becomes irrelevant which of the three commands was used, as it overrides that. The command will pass it as the "open type" to the driver (IO.OPEN D3.W).

For directory structured device drivers, this may be one of:

| | |
|---|---|
| 0 | IO.OLD (same as doing OPEN in the first place) |
| 1 | IO.SHARE (same as doing OPEN_IN) |
| 2 | IO.NEW (same as doing OPEN_NEW) |
| 3 | IO.OVERW (as TK2’s OPEN_OVER, it overwrites any existing file) |
| 4 | IO.DIR (as TK2’s OPEN_DIR, it opens the directory given) |

The other thing that uses this "open type" parameter is the "pipe_" device, where it requires the QDOS channel number of the source end of the pipe. It is possible, with some effort, to get pipes connected between MultiBASICs using "OPEN#chan,’pipe_128’:qdch=PEEK_W(\\48\\chan*40+2)" in one, getting "qdch" across to another, and doing "OPEN#chan,’pipe_’,qdch" there. See also the pipe device extensions in the Concepts chapter.

Once again, the above facilities are lost when TK2 (or anything else) comes in and replaces the calls.

PAUSE

PAUSE now takes an optional channel number, allowing you to use the procedure from programs which do not have a channel #0.

PEEK
POKE

Some improvements on these, such as allowing odd addresses, "POKE"ing lists, etc. making them much more powerful! Their syntax is now:

```
PEEK(address)                  returns unsigned byte (1 byte)
PEEK_W(address)                returns signed word (2 bytes)
PEEK_L(address)                returns signed longword (4 bytes)
POKE address[,byte]...         store a series of bytes
POKE_W address[,word]...       store a series of words
POKE_L address[,longword]...   store a series of longwords
```

where

address := absolute | \\A6offset | \A6vector\A6offset

We didn't dare enhance them by doing anything that could have been accepted by the original syntax, so latched onto the idea of omitting the first parameter. The original functionality is just enhanced to not mind the absolute address being given as an odd number, rather that rejecting is as a bad parameter, and the POKEs can have zero or more items to store. E.g. "POKE_W 131073,1,2,-1" will happily store "0,1,0,2,255,255" in the second to seventh bytes of screen memory.

The two extended forms for "address" allow you to access any job's own memory totally reliably, assuming that register A6 is pointing at it, that is! The simpler one, with both the first two parameters omitted, uses the third parameter as the offset relative to A6 to be accessed. The final form will add the longword at the offset relative to A6 given by the second parameter to the third parameter, then use that relative to A6. Clear as mud? Yes? Why bother? Well, as anyone who has ever tried to use PEEK/POKE to access SuperBASIC's table will know, you run the risk of SuperBASIC moving while you look at it. It's also pretty messy, anyway.

Some examples:

Like to save/restore the current DATA position?
`savedata=PEEK_L(\\148)` and `POKE_L\\148,savedata.`

Want the QDOS channel ID for one of your SuperBASIC channels?
`PEEK_L(\48\40*chan)` will give it.

Want to see what you've just typed?
`FOR i=0 TO PEEK_L(\\4)-PEEK_L(\\0):PRINT CHR$(PEEK(\0\i));`

Want to find out how an "INPUT a$" was ended? Look at what comes back from `PEEK(\\4\\-1)`.

## PEEK and POKE to System Variables

To read or write data from/to the System Variables or Minerva's own System Xtensions, a further enhancement to PEEK and POKE has been implemented as follows:

address= !!(System Variable offset)
!(System Vector)!Offset

Some examples:

| | |
|---|---|
| `PEEK(!124!50)>127` | will tell you when you press CTRL-ALT-SHIFT-SPACE! |
| `PEEK(!!55)` | returns NET station number |
| `POKE !!51,1` | equivalent to pressing CTRL-F5 (freeze scrn) |
| `POKE !124!51,76` | change system cursor to a green underline |
| `POKE !!136,255/0` | turn CAPSLOCK on/off |

## RANDOMISE

Minerva has implemented a more random seed to the random number generator for those who are not entirely happy with the pattern of pseudo random numbers that the standard QL generates. To switch on the enhanced random numbers, use `RANDOMISE \` and then `RND` as before.

## RENUM

We discovered some anomalous behaviour in this, and, in the process of sorting it out, made quite a few improvements. Its parameters are now exactly as per the manual, i.e. the syntax is:

**`RENUM`** `[start_line [TO end_line] ;] [first_line] [, step]`

However, it will now also permit:

**`RENUM`** `[start_line] TO [end_line] [, step]`

It now renumbers *all* the line numbers in the system, that is it now gets the current DATA line and ERLIN right. When renumbering a 1000 line program, it used to allocate over 4000 bytes of temporary space to do it, although the current requirement is approximately half of this! Not that it makes a great deal of difference, as a binary chop search is now used on this table, instead of the original sequential search. Unless you're an aficionado of "GO TO" or "GO SUB", have masses of "RESTORE"s or set up lots of "WHEN variable"s, it'll not be noticeable. We may be persuaded to vector the code, if anyone is interested in an extension function like "LOOKUP(value, integer_array)"?

AUTO and EDIT used to share all the code that RENUM used for checking its parameters, with the strange effect that you could put the "start_line" and "end_line" in, and they would be totally ignored! They now just allow the documented "[start_number] [, step]" syntax. However, in case anyone notices, for some time now "EDIT [start_number] , 0" has not been allowed.

## RESPR

It is often useful to be able to add resident procedures while there are jobs running in the QL. The new version of RESPR will, if there are jobs running, allocate space in the common heap instead. This heap will be owned by the job doing the RESPR, and will thus disappear when that job does: *DON'T* use RESPR within compiled SuperBASIC to add SuperBASIC extensions, their disappearance will confuse the system utterly. It *is* safe to add extensions to a MultiBASIC with RESPR and remove the MultiBASIC at a later point, those extensions were known only to *that* MultiBASIC.

## SBYTES SEXEC

These commands will actually accept up to seven parameters. The syntax is `device,start_addr[,length[,data_space[,extra[,type]]]]`. The defaults are all zero, except for the type in SEXEC, which defaults to 1. The "extra" parameter is what the TK2 FXTRA function returns; but unfortunately TK2 replaces SBYTES and SEXEC to get default directories working... The "type" actually allows you to set both the file type (bottom byte) and the file access key (next byte up), though this latter byte has never been used by anything except Toolkit 3, and we wouldn't know what setting it non-zero would cause.

## SCALE

You can now use a negative scale. Don't ask me why. Oh, here's a News Flash from the Technical Department...apparently it's so you can draw pictures upside down. Now I understand... Also, it removes a lot of time-wasting checks in the graphics routines.

## SELect ON

This can now use integer and string variables. Similar rules apply to these as applied to the floating point version:

```
SELect ON a$='abc'
```

will match 'AbC' as well.

If you want an exact match, you must use the:

```
SELect ON a$='abc' TO 'abc'
```

construction. As with floating point a single number need only be approximately equal (==), but a range compares from greater-than-or-exactly-equal to less-than-or-exactly-equal. This makes no difference with integers, of course!

## VER$

This has changed. Surprise. You can also give it parameters now:

| | |
|---|---|
| `VER$(-2)` | returns the base address of the System Variables |
| `VER$(-1)` | returns the current Job ID |
| `VER$(0)` | returns SuperBASIC version |
| `VER$(1)` | returns QDOS version, e.g. 1.96 |

VER$(-2), System Variables address, should *ALWAYS* be used if you feel

you *MUST* peek and poke in the SVs, and for some reason don't want to use the new PEEK and POKE syntax. Don't use it to find out how many screens you've got − we may decide to move the 2-screen system variables up to the top of available RAM. If we feel like it.

## WHEN ERRor
## WHEN variable

The WHEN keyword is used to implement a sort of "implied subroutine" system, where the programmer doesn't explicitly write a procedure call or GOSUB but lets it happen when the conditions are right, as it were.

Having said that, WHEN ERRor routines are executed when conditions are wrong, i.e. an un-trapped error has occurred. The syntax is:

```
WHEN ERRor:<statements>
```

or

```
WHEN ERRor
        <statement>
        <statement>
        <statement>
END WHEN
```

If an error occurs then the statements on the WHEN ERRor line, or between the WHEN ERRor and END WHEN lines, will be executed. Normal execution will then resume at the statement after the one that caused the error. SuperToolkit II users can use the improved `CONTINUE` and `RETRY` statements to resume elsewhere.

If an error occurs within the WHEN ERRor routines, then the program will halt with the usual error message, but with the additional information "during WHEN processing" added. The WHEN ERRor routine is added when it is encountered: thus errors in statements executed before this will cause errors as usual, and the recovery routine can be changed by passing through another WHEN ERRor block. The last such block encountered remains in force even after the program stops, so errors in the command line will cause a recovery attempt − you can turn this off by typing WHEN ERRor at the command line.

```
100 WHEN ERROR:PRINT "Whoops!"
110 PRINT "The answer isn't",1/0
120 WHEN ERROR
130   PRINT "Eeek!"
140 END WHEN
150 PRINT 1/0
```

will thus print

```
Whoops!
Eeek!
```

and all subsequent error messages will be `Eeek!` until you type WHEN ERRor at the command line!

`WHEN variable` will execute a routine when a simple variable is assigned to. It does not work with arrays, nor when a variable is `INPUT` or `READ` into.

A number of WHEN conditions can be set up for a variable, and you can of course have multiple WHEN variables. If WHEN conditions overlap there is no guarantee as to which will be chosen.(???)

```
100 WHEN i=5:PRINT "i is five"
110 WHEN i>8:PRINT "i is big"
120 FOR i=1 TO 10:PRINT i
```

will print

```
1
2
3
4
i is five
5
6
7
8
i is big
9
i is big
10
```

You can get very silly with this facility

```
100 WHEN a=1:PRINT "a is one"
110 WHEN a=2:PRINT "a is now two":b=5
120 WHEN b=5:a=1:PRINT "b is five":a=2
130 a=2
```

will print:

```
a is now two
a is one
b is five
```

and end up with a=2. Note that because the WHEN block at line 110 was already active when the last statement of line 120 is executed, is doesn't get re-entered. If you alter line 130 to `b=5`, you'll get:

```
a is one
b is five
a is now two
```

Provided the condition starts with a simple variable, it can be as complex as you like: `WHEN a>5 AND a<10` is valid.

## WINDOW

The syntax is now extended to:

```
WINDOW width, height, x, y[\ border]
```

allowing the border size and colour to be specified as the window is moved. "border" takes the form "size[, colour]" exactly as on the `BORDER` command. This facility may have to be modified in the future, as other ROMs do not

check the number of parameters, and existing software that passed such extra parameters, which were being ignored, is now causing problems. At present, Minerva is not checking for the "\" delimiter above, but this may well be the way we will circumvent the problem, by ignoring extra parameters on calls that do not use the backslash as the delimiter.

## Speed

Various improvements have been made to the execution speed of SuperBASIC. Floating point and string calculations have been speeded up, particularly concatenation of very long strings and short strings or numbers. Internal calculations stay in integer form for as long as possible, speeding things up considerably. For example, the following program runs about 40% faster on Minerva than it does on JS:

```
10 k%=1:s=DATE
15 FOR i=1 TO 5000
20   j%=k%+k%-k%*k% DIV k%&&k%||k%
30 END FOR i:PRINT DATE-s
```

Program searching, used whenever you change the flow from straight through with an `IF`, `FOR`, `REPeat`, procedure call etc., has been substantially improved. The following program:

```
100 DEFine PROCedure null
110 END DEFine
1000 REMark
1001 REMark
. . .
1998 REMark
1999 REMark
2000 FOR i=1 to 1000:null
```

runs at about twice the original speed. In passing, note that SuperBASIC doesn't reward putting procedures at the start of the program: they have to be close to the call for maximum speed.

## Graphics

Many improvements have been made in the area of graphics. The routines have been substantially re-written to improve their speed and robustness. In particular, narrow ellipses and shallow arcs no longer go haywire. These improvements carry through to any machine-code that calls the graphics TRAPs, although anything that does its own graphics will show no change.

## Strings

Previously, string slicing needed the first parameter explicitly specified − `a$(TO 10)` failed because it was converted to `a$(0 TO 10)`. Now you can use this, and you get `a$(1 TO 10)`, which was what you wanted.

You can also now slice the '' string off either end... any of you who have got irritated that you can pick characters one by one off the front of a string, but have to change when you get down to the last one... this is for you. In fact *any* expression can now be string sliced e.g:`(a$&'x')(4)` or using `DATE$` now: `DATE$()(1 TO 4)` returns the year. You can now slice sub-strings

out of DATE$ without needing to assign it into a temporary string first.

These features only apply to simple variables and expressions, not arrays.

Both string comparison and INSTR have been speeded up by about 25%. They were actually a little slower than on a JS, but are now now about 15% faster than that. JS actually uses a far different technique, and the speed comparison is very dependent on how many, and what sort of, embedded numeric values are present.

A collection of bugs have been ironed out. These all related to the handling of the end of the string and decimal points. An example of the problem comes out if you PRINT'.0'(1)='.'

## Tracing

A hook has been inserted inside the SuperBASIC interpreter to call user-defined trace code under various circumstances. The supplied file TRACE_BIN contains a simple trace and single-step routine using this. The routines supplied are as follows:

```
TRON [ #channel | \\device ] [ ; [ first line ] [ TO
      [ last line ] ] ]
```

Traces SuperBASIC execution to the specified SB channel number, or a specially opened trace channel. The information supplied is very simple, being of the form llll:sss... where llll is the line number and sss the statement number of the statement about to be executed. TRON defaults to channel #0 and line number range 1 to 32767.

```
SSTEP [ #channel | \\device ]
```

As TRON, except that having printed the line and statement number execution halts until a key is pressed.

```
TROFF
```

Stops tracing, and closes the trace channel if it was opened with the \\device parameter.

## Integer and string FOR and SELect

There is now no construct restricted to floating point. In particular, integer FOR loops are fully supported. To maintain compatibility, strings used as REPeat and FOR variables are truncated to four bytes. (Try REPeat a$: PRINT a$:a$='x'&a$ on a JS, if you don't mind crashing, then re-boot)

As we could not come up with any consistent idea of what else to do with them, string FOR ranges are limited to single characters. E.g.:

```
FOR i$ = 'xx', 'g' TO 'a' STEP CHR$(-3):PRINT i$
```

will print xx, g, d and a.

## Integer Tokens

We felt mildly depressed typing in "`i%=i%+1`", knowing full well that the "1" would be tokenised as six bytes of floating point, and when the interpreter did its stuff, it would go through the sequence of stacking the value from "i%", remembering the "+", stacking the floating point one, seeing the end of expression, finding that it had a float at the top, so the "+" had to work in F.P., having to do various shuffles to get the integer value of "i%" out from *underneath* the F.P. one, do the addition in floating point, *then* it would see that it now had to convert the F.P. result back to an integer before putting it into "i%"!

By default, MTERV now tokenises values in three formats, adding "short" and "long" integers to the original floating point. Before anyone gets excited, in this context "short" integers means values that fit in one byte and "long" is just two bytes.

Now the sequence for "`i%=i%+1`" goes: stack "i%"'s value, remember "+", stack INTEGER "1", see end of expression, add two integers on the stack, store result in "i%".

Programs using integer tokens run about 10% faster and take about 15% less space. On looking at one 800 line program, we found it saved over 5k of memory when integers were tokenised! If one had a program that consisted almost entirely of `DATA` statements of long lists of small numbers, in such a case, the space saving can get near 50%!

The two new tokens take the following form in the stored SuperBASIC program:

| | |
|---|---|
| short integer: two bytes: $89 | value (-128..127) |
| long integer: four bytes: $8A | value (-32768..32767) |

## Compilers and integer tokenisation

Being well aware that this sort of change can have *horrendous* consequences for current versions of compilers, etc., we have allocated a byte in the "Basic Variables" area for flags to control any such major deviations from the original. Having had bad experiences in the "1.6x" versions with conflicts with the "SV" stuff, we have chosen a byte that *cannot* cause any problems, as it's one that *used* to be used by "WHEN", but isn't anymore... it's at offset $D4, decimal 212, and setting its top bit will prevent the parser from using the new integer tokens. QLiberator v3.34 can now deal with integer tokenisation − as yet we have not received news of a comparable change being made to either Supercharge or Turbo.

See below for the details, but

```
POKE \\212,128
```

will set it and turn integer tokenisation off *before* loading and compiling your SuperBASIC program! To re-enable it, use "`POKE \\212,0`".

The `REPLACE` extension which appeared in QL World, was singularly unsuccessful when tried, because it didn't know how long the new tokens

were supposed to be (it got “short” right, but thought “long” would also be only two bytes... disaster!). This is somewhat inexcusable, as anyone who delved into the earlier ROMs would have found a little table (at $9062 in a JS) which says how long each token is supposed to be! (See.....we don’t just grab these thing out of the air!)

Tokens $89 and $8A were in fact already reserved for integers, although whether they were intended to be signed or not is unknown. Anyway, a quick change to one byte in the table in `REPLACE` sorted it out. P.S. `REPLACE old_name,new_name` overwrites every “old_name” with “new_name”. It’s nice for changing short names to more descriptive ones, and on Minerva, changing floating point to integer variables. For anyone else out there who uses it, beware! It’s not particularly safe. The most reliable way of using it is to `LOAD`, `REPLACE` and then `SAVE` a program.

There is one repercussion of tokenising integers within Minerva itself. `RENUM` will not renumber references to lines 1..127 in `GO TO`, `GO SUB` or `RESTORE`. This should rarely be a hardship...

One other thing that will *not* work is the recommendation in the Toolkit II manual for “`BPUT`”ing floating point. With Minerva’s integer tokenisation on the loose, “...+0” will not force floating point. You need to use something like “...+1E-555”, or have “fp=0” set up, and use “...+fp”.

## Parser

Major re-writes of the parser make it more efficient, and hence, faster.

The “bad line” cursor is now put just about slap-bang at the error: i.e. everything to the left of it, the parser made sense of, but whatever the cursor is sitting on, at that point it couldn’t puzzle out what you meant! So if you type `i=INT((i+3)/4))*4` the cursor will be placed on the last “)”.

For some obscure reason, monadic operators (-, +, `~~` and `NOT`) were restricted occurring singly. Now, this crops up very rarely, but we have been caught out by it. It’s quite amusing to have `i%=-~~i%` as an alternative for `i%=i%+1` ! Following on at this point, were you aware that using `i=-1` was actually stored with a “monadic minus” followed by a *positive* one! It’s not anymore! If you really want that to happen (and we can’t think why!), you must now type `i=-+1`.

Another interesting idea concerned slicing, in respect of function parameters. We all know that `DATE$(1 TO 4)` returns “bad parameter”, as “DATE$” gets given two parameters, instead of having the year sliced out of it, which is what you were thinking about doing. “Right!”, you say, and try to get round it with `(DATE$)(1 TO 4)`. Still no luck! JS says “bad line” now, and sticks with that when you have the brainwave of `DATE$()(1TO 4)`, (“I know! I’ll try giving it no parameters!”).

You then give in and have to do `temp$=DATE$:temp$=temp$(1 TO 4)`, or worse. Minerva fitted, and *both* ideas work! Minerva will now slice *anything* (including numbers!) and is perfectly happy with a parameter list with no parameters in it.

The interpreter now parses those instances where the original insisted that you put a space after a keyword, even though it looked as though it wouldn't need it. For example how often have you typed things like `DATA'fred','jim'`, only to have it thrown out as a bad line? Minerva's parser will now insert a space for you.

## MultiBASIC

There are very few differences between a "MultiBASIC" and the standard SuperBASIC interpreter, job 0. A MultiBASIC can be started in exactly the same way as any other job, using EXEC, a "front-end" program, or one of the QJump hotkey systems (highly recommended) − a new vector allows an EXECed job to promote itself to being a SuperBASIC interpreter. This will inherit all the procedures and functions available to its parent interpreter: any others added to the parent subsequently will not be seen by the child interpreter, and any added to the child are only seen by it and its offspring, disappearing when it is removed.

This all sounds wonderful but it must be horrendously difficult to use this MultiBASIC facility isn't it.....? Well no actually! Watch carefully, this is all there is to it

```
EXEC 'flp1_multib_exe'
```

You are now looking at another SuperBASIC interpreter which to all intents and purposes behaves just like the original with the noted exceptions below.

A MultiBASIC can have its job priority altered just like any other job and any toolkit extensions which relate to job control should work in the usual way.

Please note that you should load *only* SuperBASIC extensions into a MultiBASIC, unless you can *guarantee* that you'll *never* throw it away. Loading operating system extensions, such as QJump's Pointer Interface, is almost bound to cause problems if they disappear when the owner job goes away! Packages in the latter category include Lightning, and SuperToolkit II with the MDV extensions: SuperToolkit II without the MDV stuff, the Pointer Toolkit, and the Turbo Toolkit should be safe enough.

The MultiBASIC supplied has just one channel opened for it, which is used for both channels #0 and #1. If you want something that looks like an ordinary SuperBASIC interpreter, as seen at boot time, the following program will do the trick − note that it needs SuperToolkit II:

```
100 OPEN #0;con:OPEN #1;con:OPEN #2;con
110 WMON 4
```

## Removing a MultiBASIC

A MultiBASIC will remove itself if it encounters an error while reading a new command from its primary command channel, #0. You can therefore get it to go away by typing `CLOSE #0` at it.

For more advanced use, you can use `QX` or `EX` to pass channels and/or command string. If the last character of the command string is the "ROM" marker (an exclamation mark) it is removed from the string and the

interpreter will start up with only the original ROM names, instead of inherited names. The remaining command string is then scanned for the “file” marker (a greater-than sign), and if it’s got it, the first part is opened as an input command channel, and the rest is shuffled down.

The command string, what’s left of it, becomes `CMD$` in the interpreted MultiBASIC. Channels passed:

| | |
|---|---|
| None: | If no file marker in the command string, a single window is opened for both #0 and #1 |
| One: | Slotted in as both #0 and #1 |
| Two: | Become #0 and #1 |
| More: | First two become #0 and #1, #2 is missed out, and the rest go in as channels #3 onward. |

E.g. A filter to replace strings in a file:

```
100 a$='':i%='/'INSTR cmd$
110 IF i%:a$=cmd$(i%+1TO):cmd$=cmd$(TO i%-1)
120 c%=LEN(cmd$)
130 REPeat lp1
140  IF EOF(#0):EXIT lp1
150  INPUT#0;i$:IF c%
160   REPeat lp2
170    i%=cmd$INSTR i$:IF NOT i%:EXIT lp2
180    PRINT i$(TO i%-1);a$;:i$=i$(i%+c%TO)
190    END REPeat lp2:END IF
200  PRINT i$:END REPeat lp1
999 IF VER$(-1):POKE\\48\\0,-1
```

Save this in a file called “flp1_c_bas”, then use:

```
EX flp1_multi,flp1_in,flp1_out;'flp1_c_bas>fred/jim'
```

to convert all occurrences of “fred” in “flp1_in” to “jim”, writing the result to “flp1_out”.

A further tweak is permitted: we may even tell the new interpreter to use a specified set of machine code names by giving it a positive value in A1. This option is just a bit weird and we don’t support it at the moment. Note that MultiBASICs cannot be re-activated: the vector entry sorts this out.

## Resident MultiBASIC

An additional MultiBASIC file is now provided, *MultiB_REXT*, which when loaded with `LRESPR` or the following:

```
base=RESPR(344):LBYTES flp1_MultiB_rext,base:CALL base
```

will add the SuperBASIC extension ‘MB’ which will invoke a new copy of the MultiBASIC job. This is allows you to have MultiBASICs available without needing to `EXEC` them from a disk or microdrive but it is not quite as convenient as having them resident on a hotkey.

## Missing END DEF, ERROR and BREAK problems

It was sometimes very odd/messy when a SuperBASIC program broke out because of an error, was "CTRL[-ALT]-SPACE"d or simply dropped off the end of the program whilst still in a FuNction. Sometimes there were spurious error messages, weird "At line" reports, and even immediate re-execution of part/all of the program, especially if `EDIT` was used. I think most of these problems have now been solved.

## I²C access function I2C_IO

The SuperBASIC function `I2C_IO`, supplied on the utilities disc, uses the vector II_DRIVE, described in the Assembler chapter, almost directly:

```
result$=I2C_IO(command$,res_len[,device[,parameter]])
```

The command buffer data are contained in `command$`. The data buffer is effectively “write only”, being the result of the function, and is thus not available as a data source; in addition, the anticipated length of the result must be supplied as the second parameter so that space can be allocated to store it.

So:

```
PRINT I2C_IO(CHR$(164)&CHR$(16)&CHR$(3)&CHR$(188)&CHR$(255),4,80,1)
```

will read 4 bytes from location 16 of the RAM.

CHR$(164) is a normal I/O byte saying “write <parameter> bytes to the <device>”; the device and initial parameter are set to 80 (the 8583 chip) and 1 by the last two parameters.

The 1 byte is taken from the command stream, so that absorbs the CHR$(16) and sets the 8583’s address counter to 16.

The CHR$(3) sets the parameter to 3; this will be used as the count of bytes read AND acknowledged, so the estimated result length is 4.

The CHR$(188) then says “read <parameter> bytes and <stop>” so the four bytes from 16-19 are read.

CHR$(255) then terminates the command stream, and the four bytes read are returned as a string.

Examples of the use of this may be found in the configuration program: `hpeek$` returns `l` bytes from address `a` in the RAM, and `hpoke$` puts the string `s$` to address `a` − it’s simpler than it seems from the above example!

If you intend experimenting with this, we’d suggest you use the “save configuration to file” option in the configure program, as a wrong command string does tend to corrupt the RAM.

## RTC memory map

The 8583's memory map is as follows:

| | |
|---|---|
| 0-15 | control and clock |
| 16-255 | RAM |

The control and clock bytes should not normally be accessed from user programs: a configuration program is provided to set these correctly, and writing unexpected values here may cause Minerva to mis-interpret the clock contents.

This version of Minerva defines uses for the following RAM locations:

| | |
|---|---|
| 16-19 | expected QDOS version number, e.g. 1.89 |
| 20-23 | re-boot "D1" value |
| 24-25 | year*2+month DIV 10 |
| 26-27 | copy of locations 22 and 23 |
| 28-29 | ROM disable bits |
| 30 | NET station number |
| 31 | SX_TOE value, "System Turn Off Enhancements" |
| 32 | BV_TOE, "SuperBASIC Turn Off Enhancements" |
| 33-34 | unassigned, reserved |
| 35 | length of boot string, 0 to 128 |
| 36-163 | boot string and user area |
| 164-251 | unassigned, reserved |
| 252 | SER1 device |
| 253 | SER2 device |
| 254 | PAR device |
| 255 | unassigned, reserved |

The expected QDOS version must match the actual version of the ROM − if it does not the QL will not adopt the configuration set up in the rest of the RAM. This feature is for upward compatibility.

The re-boot D1 value is as documented for the CALL 390 warm re-start facility in the Assembler chapter − see this for a detailed explanation of the various options. Different values will allow either a full RAM test at reset or a quicker RAM clear with minimal testing, and choice of monitor or TV mode with one or two screens enabled.

The year and month are set by the configuration program, and maintained by QDOS, as the 8583 only provides enough year information to keep track of leap years.

The copy of locations 22 and 23, the low-order part of the re-boot value, is used to ensure that a completely corrupt system can be re-started. Locations 22 and 23, once read, are set to a known sensible value before the re-boot proceeds. If this fails due to 20-23 having a completely stupid value in, another reset will use this value and the QL can be started normally. The configuration program can then be run to set sensible values for this and other RAM locations.

Locations 28 and 29 contain 16 bits which can be used to disable plug-in ROMs selectively. First, note the order in which the ROM banners appear on the start up (F1/F2 etc.) screen. The first of these can be made to “disappear” by setting the top bit of location 28, the second by setting bit 6, the tenth by setting bit 6 of location 29, and so on. If your boot screen has something like:

```
CARE/QJUMP  TK2.21  © 1985
Digital Precision LIGHTNING 2.10
CST QDISC v1.18 © 1984
```

setting location 28 to 96 (01100000 binary) will map out the second two ROMs leaving you with just SuperToolkit II.

The NET station number, SX_TOE and BV_TOE are just copied to the relevant places in RAM. The net station should be obvious.

If the top bit of SX_TOE is set, then Minerva allows you to format media with open files on, which is dangerous but required if you wish to format PC discs under Conqueror. Setting the top bit of BV_TOE disables the tokenising of integers, which will make SuperBASIC run slower but will prevent old versions of QLiberator and all current versions of Supercharge and Turbo from getting confused.

The boot string is inserted into SuperBASIC channel #0 as if it had been typed at the keyboard. A length of 0 implies you don’t want a boot string.

So a boot string of:

```
F1 TK2_EXT:LRUN 'N3_WIN1_REMOTE_BOOT'
```

will "press F1", enable SuperToolkit II and then run a boot file from a networked hard disc.

All “unassigned, reserved” areas are just that: QView reserves these locations for future enhancements to Minerva or for use by third party hardware or software. All allocations to third party products must be made via QView so that clashes can be avoided.

Any spare space left over after you’ve set up your boot string may be used for your own purposes, and will not be allocated for new features in future versions of Minerva or for third party products.

The SER1, SER2 and PAR devices are not used directly by Minerva. They are for use by any software that uses printers or modems, so that programs can find out if such items are present and what type they are.

Currently defined values are:

| | |
|---|---|
| 0 | nothing connected to this port |
| 1 upwards | printer type code as used by Tony Tebby's SDUMP routines e.g. 1=Epson MX80, 8=Epson LQ2500 colour,18=Brother 8056 etc. |
| 253 | Tandata modem |
| 254 | Astracom "native" modem |
| 255 | Astracom/other Hayes-compatible modem |

We anticipate that new printer types will be added at the low number end going upwards, and modems from the high numbers down.

# Assembler

WARNING: this section is for the real hackers amongst you! If you're not well into machine code programming for the QL, a lot of this will mean very little to you. That's not to say it's not worth reading it, it's all good stuff, but don't worry if you find it incomprehensible.

## Start up

Some new features have been added to the start-up sequence, and a clean way to re-boot added.

The QL may now be re-booted by calling the reset code at address $186 directly:

```
MOVEQ   #0,D1     ; pretend we hit the button
JMP     $186      ; re-boot machine
```

You can also use this facility from SuperBASIC, thus:

```
CALL 390,d1_value
```

Different values in D1 will cause various aspects of the initialisation to be skipped:

| Bit to set | Value to add | Effect |
|---|---|---|
| 0 | 1 | skip test, just clear memory to 0 |
| 1 | 2 | skip ROM scanning |
| 2 | 4 | use memory limit in bits 14..31 |
| 3 | 8 | default to TV mode |
| 4 | 16 | don't wait for F1..F4 |
| 7 | 128 | enable dual screens |
| 8..13 | n*256 | leave n*64K between screen and SVs |
| 14..31 | n*16384 | "cut" RAMTOP to a multiple of 16K |

Unused bits are reserved, and should be set to 0.

BEWARE! CALL 390 *can* crash the machine − if you tell it to allocate too many blocks of 64K + second screen, if bigger than the upper limit of memory in the machine it will push your system variables off the top of physical memory which naturally will upset it!! Similarly attempts to reduce RAM size below 48K are doomed to fail!

NOTE: ROM scanning *will* take place despite a CALL saying No Roms Today Please if the timeout on F1/F2 occurs *or* you attempt to start up in a mode from the keyboard which is not compatible with the CALL.

So, to cut RAM to 128K and omit the ROM scanning, thus giving you a totally un-expanded machine, you can

```
CALL 390,(128+128)*1024+4+2
```

To do the same, and enter TV mode automatically,

```
CALL 390,128*1024+16+8+4+2
```

A reset can also be invoked from the keyboard at any time by pressing CTRL-ALT-SHIFT-TAB.

## ROMs

The ROM scanning has been extended: the range covered is now $C000, $10000, $14000, and from the top of RAM (as indicated by SV_RAMT) upwards. This allows use of a machine with, say, 256K of RAM and 256K of ROM in the RAM expansion area.

## Exceptions

On perusing the ROM we found that the vectors that are used when instructions of the form Axxx and Fxxx are encountered (the Line 1010 and Line 1111 emulators, as Motorola call them), have BSR instructions in them. This means that any software which goes haywire and tries to execute such an instruction will leap off to a really stupid place and probably crash the machine. We've therefore made such instructions follow the same path as the illegal instruction exception, so they're trapped by QJUMP's QMON and other machine code monitors, or you can even use them.

## Traps

The TRAP and scheduler entries have been speeded up, so there are lower overheads on system calls in a machine full of jobs. One consequence of this is that the job and channel tables are scanned from the end backwards, rather than from the start forwards as was previously the case − we can't conceive of a way in which you could fall foul of this, but you never know...

In an attempt to make life easier while debugging machine-code programs, we've modified the effect of TRAP #0 slightly. It still enters supervisor mode, but now does so without smashing the trace flag so you don't need to have enabled supervisor mode tracing to trace through your own supervisor code. TRAPs of other numbers are unaffected, and won't be traced unless you turn on supervisor tracing explicitly.

In this area, we've made it safe to execute several QLiberated jobs in quick succession (e.g. from QRAM or another front-end). Previously the second (and subsequent) QLiberated jobs would cause SuperBASIC to move while the first one was still looking at SuperBASIC's name table, usually causing a spectacular crash. For all we know, this happens with Supercharge and Turbo too.

In this area, there is a problem with machine-code extensions to SuperBASIC which need more space than is available on the RI stack. The system only knows how to add space to an *interpreter's* stack, so if this happens in a compiled program it won't get the expected space increase, and will either crash spectacularly or merely behave peculiarly. We have allowed for this by ensuring that any calls to BV.CHRIX by a job other than the interpreter which result in a requirement to increase the size of the RI stack, cause the offending job to be force removed from the machine.

## RI Vectors

While improving the graphics, it became clear that a number of useful functions should be added to the arithmetic operations vector. The new operations are odd positive numbers, as follows:

| | | | | |
|---|---|---|---|---|
| RI.ONE | equ | $01 | -6 | push constant one |
| RI.ZERO | equ | $03 | -6 | push constant zero |
| RI.N | equ | $05 | -6 | followed by a signed byte, to push FP -128 to 127 |
| RI.K | equ | $07 | -6 | plus a byte, nibbles select mantissa and adjust exponent. Following byte values may be: |
| | | | | RI.PI180 equ $56 |
| | | | | RI.LOGE equ $69 |
| | | | | RI.PI6 equ $79 |
| | | | | RI.LN2 equ $88-$100 |
| | | | | RI.SQRT3 equ $98-$100 |
| | | | | RI.PI equ $A8-$100 |
| | | | | RI.PI2 equ $A7-$100 |
| RI.FLONG | equ | $09 | -2 | float a long integer |
| RI.HALVE | equ | $0D | 0 | TOS / 2 |
| RI.DOUBL | equ | $0F | 0 | TOS * 2 |
| RI.RECIP | equ | $11 | 0 | 1 / TOS |
| RI.ROLL | equ | $13 | 0 | (TOS)B, C, A → (TOS)A, B, C (roll 3rd to top) |
| RI.OVER | equ | $15 | -6 | NOS |
| RI.SWAP | equ | $17 | 0 | NOS ↔ TOS |
| RI.ARG | equ | $25 | +6 | arg(TOS,NOS)=a, solves TOS = k*cos(a) and NOS = k*sin(a) |
| RI.MOD | equ | $27 | +6 | sqrt(TOS^2 + NOS^2) |
| RI.SQUAR | equ | $29 | 0 | TOS * TOS |
| RI.POWER | equ | $2F | +2 | NOS ^ TOS, where TOS is a signed short integer |

## MT.BAUD

As well as the standard baud rates (75, 600, 1200, 2400, 4800, 9600 and 19200), the value passed in D1.W may be given as a code to control independent output baud rates. The additional values accepted are small negative numbers, in the range -128 to -1. As a bit pattern, it is:

| Bit | Add | Function |
|---|---|---|
| 0..2 | 0..7 | baud rate − 0=19200, 1=9600,...7=75 baud |
| 3 | − | Always 0 |
| 4 | 16 | Set to force this port to ignore ordinary BAUD calls |
| 5 | − | Always 0 |
| 6 | 64 | Set to signal SER2. else SER1 |
| 7..15 | -128 | Signals new format baud control word |

Bit 1 of SX_TOE may be set to disallow these extended commands.

The two bytes at SV_TIMOV are now used to control the two serial ports independently. Their five least significant bits hold the bits passed in bits 4..0 by the above extended commands, or, if their bit 4 is not set, their least significant three bits are allowed to be set by the normal command values.

The upper three bits are used to keep track of the status of transfers with the respective ports. They are not well defined for old 8049s. For Hermes, bit 7 set means that data bytes have been lost, because no handshake is in use and Hermes was still getting characters after its 32 byte buffer was full. Bit 6 indicates that framing errors have occurred. A framing error is when the stop bit of a byte is not seen as a zero. These may mean the wrong baud rate is selected, or that these is "line noise". Bit 5 is not yet defined.

Using an extended command will always return a value in D1.B, comprised of the original value in the control byte. The command will also reset bits 7..5 of the control byte to zero.

N.B. If either input line is required, and Hermes is *not* fitted, DO NOT use these commands, as the serial handling will just become confused. Only if *neither* input line is required is it permissible to operate the two output lines at differing baud rates.

## MT.DMODE

This requires a section all of its own, as it's become pretty complex. The original values of -1, 0 or 8 for D1 and -1, 0, 1 or 2 for D2 still apply, so existing programs will still work as expected.

The enhanced set of options now available needs to operate on up to six bits of information. These are the 4/8 colour and visible states of each of two screens (4 bits), which screen is currently displayed and which screen is the current default for this job. They may all be read, *en masse*, but there is a need to be able to set them selectively, which means 12+ bits!

In order to maintain compatibility, even with people who send the wrong parameters, the new options have bits 7/6 of D1.B differing.

The primary needs are to be able to change the default screen and make it blank or visible. Being able to force which screen is on display is rather undesirable for too many programs to try doing simultaneously!

With these in mind, we define the options available from D1 bits as follows:

| D1 bit | Effect |
|---|---|
| 0 | visible/blank on other screen |
| 1 | visible/blank on default screen MC..BLNK |
| 2 | mode4/mode8 on other screen |
| 3 | mode4/mode8 on default screen MC..M256 |
| 4 | display scr0/scr1 |
| 5 | default scr0/scr1 (N.B. takes effect BEFORE all other options) |
| 6 | Clear − use D1.W. Set − use D1.B (same as D1.W msb all ones). |
| 7 | opposite to bit six, i.e. 7/6 = 0/1 or 1/0 always |
| 8-15 | Ignored if bit 6 set. Otherwise ... |
| 8-13 | Associated with bits 0-5 (clear=absolute, set=toggle) |
| 14 | Clear − force redraw of other screen |
| 15 | Clear − force redraw of default screen |

If bits 6 and “x” + 8 are clear, use bit “x” to force absolute selection. Otherwise, toggle settings as per bit “x”. (“x” = 0..5) If bits 6 and “x” are clear, invoke that screen redraw. (“x” = 14..15) The original -1/0/8 options are equivalent to $40 (or -128), $7780 and $7788.

The “default screen” is initially inherited from the parent job. It is the screen on which newly opened con/scr channels will appear. It is also the screen affected/reported on by D1.B = 0/8/-1. A change to this takes effect before the rest of the above is looked at. It then remains changed for the job, until the job does another change to it.

The “mode4/mode8” selection in this extended system does *not* operate the same as the basic D1.B = 0/8 options. It *only* changes the physical display mode. Redrawing of the windows is independent. It is also not forced by D2.B being positive, except on D1 = 0/8/-1 calls.

The “displayed screen” is the one currently on display.

The least significant bit of the JB_RELA6 variable in a jobs header is where its “default screen” is recorded. 0=scr0, 1=scr1.

The returned value in D1.B for the extended calls is as follows:

| bit | 0 | 1 | which screen | |
|---|---|---|---|---|
| 0 | visible | blank | other | |
| 1 | visible | blank | default | |
| 2 | mode4 | mode8 | other | |
| 3 | mode4 | mode8 | default | MC..M256 |
| 4 | scr0 | scr1 | display | |
| 5 | scr0 | scr1 | default | |
| 6 | | | | |
| 7 | single | dual | available | MC..SCRN |

## SuperBASIC trace

A “hook” has been added to SuperBASIC to allow user-supplied trace routines to be added.

```
BV_UPROC   equ $7C    ;long top-bit-set address of user trace routine
```

It may be set by something similar to the following code:

```
TST.B   BV_UPROC(A6)       ; is there already a
                             trace?
BMI.S   WHOOPS             ; better not blat it!
LEA     TRACE(PC),A0       ; point to routine
MOVE.L  A0,BV_UPROC(A6)    ; fill in its address...
TAS     BV_UPROC(A6)       ; ...with the top bit set
```

The routine is called under various circumstances, with a long word on the stack to indicate the reason:

```
TRC.STST    equ 0      ; start of statement
TRC.LET     equ 2      ; variable assignment
TRC.SWAP    equ 4      ; variable/LOCal swap
TRC.MCFN    equ 6      ; entry into machine-code function
TRC.RENM    equ 8      ; RENUM just happened
```

The user-supplied routine must preserve all registers. It should pop the reason code from the stack, perform any action suggested by the reason, and return to the interpreter with an RTS instruction. An example of a simple trace and single-step routine is supplied on this medium for you to study. We haven’t explored the full implications of a decent SuperBASIC debugger yet, but the functions above should be enough to keep track of variables and program execution. Anyone who’s interested in writing something good should get in touch with us − anyone else should treat the above with caution, it’s very much an “alpha test” facility and isn’t necessarily cast in stone yet!

## MT.RERES

This now works...

IO.EDLIN

The editing TRAPs IO.EDLIN and IO.FLINE, which are called by EDIT, AUTO and INPUT, can now accept enhanced movement keys which are as follows:

| | |
|---|---|
| ALT ←/→ | move to start/end of current line |
| TAB | move along to 8th character from start of buffer |
| SHIFT-TAB | moves BACK in same steps as above |
| CTRL-ALT ← | deletes to start of current visible line |
| CTRL-ALT → | delete from current character to total end of line |
| ESCape | behaves like CTRL/SPACE (Break) |
| SHIFT-ENTER | behaves pretty much like ENTER |
| SHIFT-SPACE | behaves pretty much like SPACE |

See the Concepts chapter for a more detailed discussion.

BV.CHxxx

The BV.CHxxx routines check space in relevant memory areas and, if necessary, allocate more. The schemes go like this:

1) A call that already has enough space available should return as fast as can possibly be arranged.
2) If a call is not immediately satisfied, the *extra* amount required is rounded up a little, and this amount is looked for in the central free area. If found, the involved sections are shuffled by this amount.
3) In extremis, the amount that the central area fell short by is requested from the system by an ALBAS trap. This will tell us how much extra we got, which may have been rounded up a bit, e.g. to a multiple of 512. The requested amount is added to the original place and any spare is given over into the central area.
4) If we can't get enough memory off the system, we trundle off to the return address held in BV_SSSAV, hopefully to report the problem.

Actually, there is *no* requirement to round requests, other than ensuring that any eventual shuffling of the areas moves them a sufficient *even* distance. The rounding happens to be convenient, as the headroom (which is desirable!) can be applied with an "addq" in the code. This code now *only* moves the active parts of each section.

A serious flaw in the original code was that once it needed to move anything, it insisted on finding the *originally* requested amount, plus headroom, plus rounding. This meant that, for instance, if a large array was re-dimensioned a little larger, although a mass of VV area might have been fully released, an "out of memory" could be reported when there was no real problem! This code now only goes for the, slightly bumped up, *extra* space needed. One other point that could be made here is that it would be very nice if the entry points (or due to history and silly software, a new set) were added, that not only checked for the requested amount, but actually updated the pointer involved!

Note: Some of these routines are expected to be at fixed offsets from BV_CHRIX by silly software.

## BV.NAME

This is used at startup of a new copy of SuperBASIC, to acquire names from its interpreting ancestor, the ROM or even elsewhere. In the interest of making the process fairly simple, though we could be terribly subtle, we get space for the whole of the Name Table and Name List, then compact it as we copy it across. We could go to the bother of doing two scans, the first just to establish the size, but it seems a bit pedantic, as the extra space can be immediately released to the central area, and it won't be vast usually. We mustn't confuse people when moving names, so this is all done in supervisor mode.

## Device Drivers

The routine which finds driver information from a filename now imposes the following two constraints on directory device driver names, failing which, they will always result in ERR.NF.

Firstly, they must be longer than one character (and less than 32768!). The exclusion of zero length names is needed as NFS_USE in TK2 "hides" its DD entry as a zero length name when it is not in use.

Secondly, they must have bit 5 clear in all bytes. This excludes lowercase 'a' to 'z', uppercase '`' to '+', digits and various other characters. A digit in the range '1' to '8' after the first character of the caller's name will invariably be expected to mark the exact end of the DD name part, and must be followed by an underscore. This syntax does allow in a few obscurities, but so far nobody has tried to produce a driver whose name contains anything but uppercase 'A' to 'Z'.

The original code had various bugs, including the fact that if one driver on the list was preceded by one with a shorter name, it was never found! This bug was detected by Tony Tebby, when he got the network driver in front of another driver starting with an 'N'.

No longer changes the update date on microdrive files. This works for anything that uses this trap, e.g. the TK2 RENAME procedure.

For those of you who don't know what the parameters are, D0 is $4A, D3 is the timeout, A0 is the channel ID of the file to be renamed, opened for write access, and A1 is the pointer to the new name, including the "mdvn_" and prefixed in the normal way with a word for the string length.

While here, I'll add the point that the FS.TRUNC trap #3 exists, D0 is $4B, D3 and A0 as per FS.RENAM but A1 is irrelevant. It truncates a file to the current position, discarding any data beyond that point.

Open overwrite of a file (IO.OVERW=3) is implemented, and has the same effect as IO.NEW if the file doesn't exist or, if it already exists, it has the effect of IO.OLD followed immediately by FS.TRUNC.

## New vectors

All the vectors described below need to be offset by $4000 to give the call address, e.g. to call UT_INSTR.:

```
MOVE.W  $13C,A2
JSR     $4000(A2)
```

Unless explicitly stated otherwise, register values are preserved.

## UT.ISTR $13C

**UT.ISTR** $13C Do an INSTR operation.

| | | |
|---|---|---|
| D0 | | 0 |
| D1 | | match offset |
| D2 | | smashed |
| D3 | | smashed |
| D7 | | smashed |
| A0 | string to search | preserved |
| A1 | string to look for | preserved |
| A6 | base address | preserved |

The string at 0(A6,A1.L) is searched, looking for a sub-string that is type 3 equal to the string at 0(A6,A0.L). Each string is in the standard format of having the first word recording the string length. The returned value in D1.L is zero if no match is found, or the offset, plus one, in the searched string where the other string has been found.

## GO.NEW $13E

**GO.NEW** $13E Do a NEW command.

| | | |
|---|---|---|
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

most registers destroyed

Clears out SuperBASIC program/variables/channels, etc.

## BP.CHAN $140 BP.CHAND $142

**BP.CHAN** $140 Get optional channel parameter, default #1.
**BP.CHAND** $142 Ditto, with a supplied default channel number.

| | | |
|---|---|---|
| D1 | channel number | preserved |
| A0 | | channel id if one exists |
| A2 | | position of channel block |
| A3 | parameter offset | updated if chan given |
| A5 | top parameter | preserved |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Errors: ERR.NO if the channel is not open.

Determines whether or not there is at least one parameter, and if it is preceded by a hash (#) sign, expects it to be the integer channel number. If there are no parameters, or the first is not preceded by a hash, then the

default is used as the channel number. The default is passed in D1.W to BP.CHAND, or will be set to the standard listing channel (#1) by BP.CHAN.

**BP.CHNID** $144 Look up a channel number.

BP.CHNID
$144

| | | |
|---|---|---|
| D1 | channel number | preserved |
| A0 | | channel ID |
| A2 | | SuperBASIC channel location |
| A6 | base of SuperBASIC | preserved |

Errors: ERR.NO if the channel is not open. In this case, the value input in A0 is preserved. Also, if the "X" flag is not set on return, the pointer in A2 is *above* the current top of the channel table.

**BP.CHNEW** $146 Start up a new SuperBASIC channel number.

BP.CHNEW
$146

| | | |
|---|---|---|
| D1 | channel number | preserved |
| A0 | channel id | preserved |
| A2 | | SuperBASIC channel location |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Errors: ERR.EX if channel is already open.

If the channel was already open, nothing will have been changed. If the channel table is extended, it is done with all $FF's. If a good slot is found, the new ID will be stored and the rest zero, except for filling in 80 as the line width.

**BP.FNAME** $148 Get a file name parameter, with or without quotes.

BP.FNAME
$148

| | | |
|---|---|---|
| A1 | | RI pointer to file name string |
| A3 | NT parameter pointer | usually updated by 8 |
| A5 | top of NT parameters | preserved |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Errors: various

This will accept a parameter suitable as a file name string. The string is put at the top of the RI stack.

## CA.CNVRT $14A

**CA.CNVRT** $14A Convert data type.

| | | |
|---|---|---|
| D0 | type required | error code |
| A1 | RI stack pointer | updated |
| A5 | top of NT stack | preserved |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Errors: ERR.XP if the conversion fails, but note that the top of the RI stack will always be left with a value of the requested type.

The item described by the top entry on the name table is converted to the requested type. The name table entry should be an internal type, i.e. it is already on the top of the RI stack, and has its type in the 4 lsbs at \endash 7(A6,A5.L) as 0 or 1 for string, 2 for F.P. or 3 for 2-byte integer. The requested type may be 1, 2 or 3. A requested type of 4 is also accepted, which will finish up as type 3, but only the logical true (1) or false (0) value will ever be left. This request will convert strings to float first, then the float, or an original integer, is tested for non-zero.

## CA.OPEXE $14C

**CA.OPEXE** $14C Execute operator.

| | | |
|---|---|---|
| D0 | operation code | error code |
| A1 | RI stack pointer | updated |
| A5 | top of NT stack | updated |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Execute a monadic or dyadic operator on the top elements of the RI/NT stacks.

Details omitted at present...

## CA.EVAL $14E

**CA.EVAL** $14E Evaluate top element of NT/RI.

| | | |
|---|---|---|
| A1 | RI stack pointer | updated |
| A5 | top of NT stack | preserved |
| A6 | base of SuperBASIC | updated |
| A7 | SuperBASIC stack | updated |

Evaluate the top element of the NT stack leaving it as an internal (RI) value.

Details omitted at present...

**IP.KBRD** $150 Action KEYROW type info.

IP.KBRD
$150

| | | |
|---|---|---|
| D0 | | smashed |
| D1 | keyrow data | smashed |
| D2 | shift key data | smashed |
| D6 | | smashed |
| A0 | | smashed |
| A1 | | smashed |
| A2 | key queue addr | preserved |
| A3 | | smashed |
| A4 | | smashed |

Main keyboard read routine. Can be called by replacement keyboard code. Must be called in supervisor mode.

The value supplied in D1 is as follows:

| **bits** | **function** |
|---|---|
| 31..6 | ignored |
| 5..3 | 7 - row number (as per KEYROW command) |
| 2..0 | bit number |

The value supplied in D2 is as follows:

| **bits** | **function** |
|---|---|
| 31..3 | ignored |
| 2 | CTRL |
| 1 | SHIFT |
| 0 | ALT |

These finish by pushing a character, possibly preceded by ALT, (CHR$(255)), into the supplied queue. This also becomes the auto-repeating character (pair). Special codes corresponding to space, TAB or ENTER keys in conjunction with CTRL and also optionally with ALT and/or SHIFT are handled, causing all sorts of wonderful things to happen.

**IP.KBEND** $152 Finalise keyboard read.

IP.KBEND
$152

| | | |
|---|---|---|
| D0-D2 | | smashed |
| D3 | no. of polls missed | preserved |
| D5 | flag if last key held | smashed |
| A2 | keyboard queue | preserved |
| A3 | | smashed |

Finish off keyboard read and handle auto-repeat.

This should be called by replacement keyboard code when all required KEYROW type data has been passed using IP.KBENC, to indicate if the last key is to be treated as still held down. Only bit 4 of D5 is relevant, and a zero means the final key has been released.

## SB.START $154

**SB.START** $154 Start a MultiBASIC.

| | |
|---|---|
| A0 | command channel ID |
| A5 | size of available area |
| A6 | base address |
| A7 | stack pointer |

This entry should be JMP'ed to. It is used to start a MultiBASIC job. The second word in the job's program space must be set to the offset from the start of the program's area to the start of the SuperBASIC tables given in A6. The value in A6 should be the base of the area available to the job and A5 is the total size available, i.e. -2(A6,A5.L) is the last word.

A7 points to the EXEC type parameters, as in:

| | |
|---|---|
| number of channels | (word), |
| channel ID's | (longwords) |
| command string length | (word) |
| contents | (bytes) |

A variable "CMD$" is preset with the command string.

If any channel IDs are passed on the stack, they are set up as the new MultiBASIC's channels #0, #1, #3, #4, etc. *Note that #2 is skipped.*

If only a single channel ID is passed on the stack, it will not only be put in as #0, but also as #1.

If A0 is zero and no further channels are given, the default monitor/TV windows are opened for #0, #1 and #2.

If A0 is non-zero, the job starts interpreting lines from the supplied source, until such time as EOF is signalled, or any other problem crops up, at which point it will close that channel and continue trying to read from #0.

**RESERVED** $156 (currently preset to -1, just for fun...).

## Move memory

These vectors are designed to allow you to move memory as quickly as possible!

The call sequence is kept as simple as possible to avoid having lots of complex code in the calling routines. To this end, it preserves the values of *all* registers, including the call parameters, except that D0.L is returned as zero.

The main routine is an absolute move, but three additional entry points provide for switching to supervisor mode and handling source and/or destination as A6 relative addresses.

The move is *always* non-destructive, i.e. if the source and destination do overlap, the move will start from the top of the areas.

A bit of a waste on a normal QL, but in order to provide for the astounding Medusa spec, we leave the code so it can actually trundle more than a megabyte around! After all, it only costs 24 bytes of code! The choice of instruction sequences to achieve the move is based on *very* exhaustive testing of alternatives. The only improvement would be to employ “movem.l”, at some considerable expense for a very minor speed increase.

This version is inefficient when called to move trivial (<256?) amounts of memory, but current callers are usually asking for much more, or have already spent so much time setting up for this that it probably doesn’t matter. With normal (slow) internal RAM, the overhead seems to be around thirty bytes worth of move. It’s a moot point whether string copying should use this routine, though I believe that it should, as it has already done a vast amount of processing before deciding to move a string, that one might just as well get the speed improvement for very long strings, without hassle.

Using “movep” instructions is a little faster than eight single byte moves on standard memory, and even quicker on faster memory.

MM.MOVE
$158

**MM.MOVE** $158 Fast memory move.

| | | |
|---|---|---|
| D0 | | 0 |
| D1 | length | preserved |
| A0 | destination pointer | preserved |
| A1 | source pointer | preserved |

The D1.L bytes from (A1) are moved to (A0). No constraints are imposed: i.e. any of D1, A0 and A1 may be odd. The only slight “get-out” is that nothing is moved if D1 is negative. This may be called in either user or supervisor mode.

The move is always non-destructive, i.e. should the source be at a lower address in memory than the destination (A1<A0), and the areas overlap (A1+D1>A0), then, and *only then*, the memory is moved starting at the top of the areas.

The move is *fast*, approaching within about 5% of the maximum theoretic speed that memory can *ever* be moved around. (The fastest move would be something like having a piece of code with enough MOVEM or MOVEP instructions to do the whole move without a loop!)

This vector can be called from within SuperBASIC using:

```
mm_move=peek_w(344):mm_move=mm_move+16384
CALL mm_move,length,2,3,4,5,6,7,destination,source
```

MM.MRTOA $15A
MM.MATOR $15C
MM.MRTOR $15E

**MM.MRTOA** $15A Fast move, relative to absolute.
**MM.MATOR** $15C Fast move, absolute to relative.
**MM.MRTOR** $15E Fast move, relative to relative.

| | | |
|---|---|---|
| D0 | | 0 |
| D1 | length | preserved |
| A0 | destination pointer | preserved |
| A1 | source pointer | preserved |
| A6 | base address | preserved |

This should *only* be called in user mode. It switches to supervisor mode in order to convert the A6-relative pointers to absolute pointers. The source is 0(A6,A1.L) for MM.MRTOA and MM.MRTOR and the destination is 0(A6,A0.L) for MM.ATOR and MM.RTOR. Once these considerations have been taken into effect, MM.MOVE is called to do the actual moving.

These two vectors will clear memory 32 bytes at a time, to give just about the fastest possible method of zeroing out memory areas. Minerva uses them for clearing common heap areas and initialising `DIM` arrays.

SS.WSER $160

**SS.WSER** **$160** **Wait for serial transmit to finish**

| | | |
|---|---|---|
| D0 | new mode | smashed |
| A6 | system variables | preserved |

Call this in supervisor mode, with interrupts enabled, to wait for any pending serial port transmission to finish. The value in D0.B should be one of PC.MDVMD ($10) or PC.NETMD ($18).

SS.RSER $162

**SS.RSER** **$162** **Re-enable serial transmission**

| | | |
|---|---|---|
| A6 | system variables | preserved |

Call this in supervisor mode to re-enable serial transmission.

MD.SELEC $164

**MD.SELEC** **$164** **Select microdrive unit**

| | | |
|---|---|---|
| D0 | | smashed |
| D1 | drive number 1..8 | smashed |
| D2 | | smashed |
| A3 | $18020 | preserved |

Call this in supervisor mode, with interrupts disabled, to select which of the microdrive units is to be started up. (N.B. D1.W should be 1..8). A delay of about half a second after this call should be sufficient to ensure that the drive is up to speed.

**MD.DESEL $166** **Deselect microdrives**

MD.DESEL $166

| | | |
|---|---|---|
| D0 | | smashed |
| D1 | | smashed |
| D2 | | smashed |
| A3 | $18020 | preserved |

Call this in supervisor mode, with interrupts disabled, to stop microdrives.

**MM.CLEAR $168** **Fast memory clear**

MM.CLEAR $168

| | | |
|---|---|---|
| D0 | | 0 |
| D1 | length | preserved |
| A0 | destination pointer | preserved |

The D1.L bytes at (A0) are set to zero. D1 and/or A0 may be odd. If D1.L is negative, nothing is touched.

**MM.CLRR $16A** **Fast clear, relative**

MM.CLRR $16A

| | | |
|---|---|---|
| D0 | | 0 |
| D1 | length | preserved |
| A0 | destination pointer | preserved |
| A6 | base address | preserved |

This should only be called in user mode, as it switches to supervisor mode and back, to speed up the operation. The D1.L bytes at 0(A6,A0.L) are set to zero. D1 and/or A0 may be odd. If D1.L is negative, nothing is touched.

**IO.QSETL $16C** **Set up long word queue**

IO.QSETL $16A

Allows queues longer than 32766 bytes to be set up.

The call is identical to IO.QSET, except that it genuinely expects the queue length to be the full longword in D1. Previously, only the least significant word in D1 was used, and disastrously so, should it be negative!

Note that the value passed in D1 (in both cases) must be one more than the maximum number of characters to be held in the queue.

When a queue of more than 32767 characters is in use, it is possible that callers might not like being told a value higher than this when the vector IO.QTEST is called. The Technical Guide does not actually specify whether the value returned is supposed to be the whole of D2.L or just the least significant word of it. To avoid any possible problems, IO.QTEST limits itself to a response of 32767.

**II_DRIVE $172 Execute I²C transaction**

| | | |
|---|---|---|
| D0 | | error code |
| D1 | | register result |
| D2 | device \| parameter | smashed |
| A1 | pointer to data buffer | updated |
| A3 | pointer to command buffer | updated |

Each device on the I²C bus has a device address which is seven bits long: that of the 8583 in this system is set to 80. A complete bus transaction consists of a start condition followed by a number of byte reads or writes, terminated by a stop condition. Every byte read or written during a transaction will be acknowledged by its recipient, except (usually) the last. The first byte after the start contains the address of the desired device, plus one bit designating the transaction as a read or write: the content and direction of subsequent bytes depend on the device addressed.

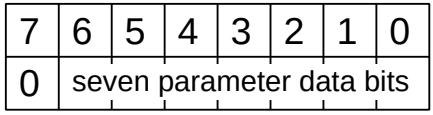
The 8583 has 256 locations that can be read or written. Thus the first action required is a write to set up the address to be accessed. To simplify matters, once this address has been accessed the 8583's internal address counter increments, so that multiple consecutive locations can be accessed without re-writing the address explicitly. If write access to the location is what is wanted, then the data byte(s) can follow directly after the address. If read is required, then a new start+device must be sent, with the read/write set to read.

The I²C driver vector is controlled by a byte stream contained in the (read-only) command buffer. Data to be written may come from either the command or data buffer. Results may be returned into D1 or the data buffer.

Four error codes are returned at present: ERR.FF implies that the hardware is not functioning: ERR.NF that the addressed device is not present: ERR.TE that an acknowledge was not received when it was expected: and ERR.BP that a bad command byte has been encountered.

During the interpretation of the command stream, D2 holds the number of the addressed device in its more significant word, and a "parameter" word in its less significant word.

Command stream bytes are as follows:
Parameter build byte:




The contents of the parameter word are shifted left seven bits, and this byte is "OR"'ed into it. A contiguous sequence of three of these can be used to set up a full 16 bits of parameter. Only two uses of this are currently made. A single byte is used before a special command which is to copy it to the device group register, so we can change devices during a sequence. The other usage is to set up the byte count for a normal I/O command. This will

make use of a 16-bit count, and may need anything from zero to three of these parameter build bytes. The parameter register is always cleared to zero after each of the normal I/O and special byte types has been processed.

Normal input/output byte:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | S | R | B | P | A | 0 |

The bits of this byte are essentially handled from left to right, to allow the most typical I/O sequence to be handled in its entirety.

S = 0: no START required (assumed SDA high and SCL low)
S = 1: send START and device (SDA/SCL assumed. high)
R = 0: write mode, or R = 1: read mode
B = 0: if R=0, write from control, or R=1 read to register
B = 1: write/read uses data buffer
P = 1: send STOP sequence
A = 1: send acknowledge on last read (R=1) byte

R=0 and A=1 is invalid, as is R=1, P=1 and A=1. Also bit 0 must be clear. If these conditions are not met, an err.bp is reported after processing all but the P bit.

The parameter value specifies the exact byte count for a write sequence, but on a read (R=1) sequence, it counts only those bytes to be acknowledged. If R=1 and A=0, the final byte with standard non-acknowledge is extra.

Write sequence data byte:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| data byte | | | | | | | |

If a normal I/O byte requests writes from this control buffer, it will be immediately followed by the appropriate number of data bytes to be written.

Special I/O and control byte:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | G | V | D | C | 1 | Q |

Once again, the bits are handled from left to right, and these control all the exceptional cases we wish to cope with. Note that the SDA and SCL setting will occur simultaneously, hence to be valid, only one should differ from its currently known state. If V=0, the state will always be both ones before they are applied, so the combination of V, D and S all zero is always invalid.

G = 0: set device group addresses as 2 * current parameter value
G = 1: assume device group is already in its register
V = 0: kill bus (assume NOTHING about bus, ensure in standard free state)
V = 1: assume the bus is valid, whatever state it is in
D = d: set SDA
C = c: set SCL
Q = quit

Note that bit 1 is reserved and must be set, or an err.bp is reported after processing the G and V bits, but before setting the D/C combination

The control buffer must finish up with a special command that has its quit (lsb) set. Normally this will be all ones, but where the bus is not being released between calls a value of $F3, keeping SDA and SCL low, will be typical.

The general rules for the bus go as follows:

Before a START+device, SDA should be high. After a START+device, SDA will be high and SCL will be low. For a read/write, SDA high and SCL low are required and are left the same. Before a STOP, SCL low is expected and both SDA and SCL will be left high. Before an initialise, SDA and SCL are irrelevant. When using the "special" command, only one of SDA and SCL should be changed at one time. When it includes an initialise, that will preset them high.

Thus to write $1234 to location $56, the following is sent:

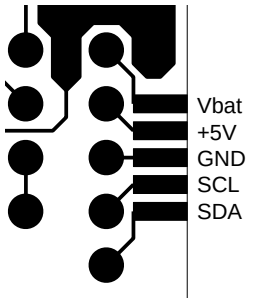
<start><$A0><$56><$12><$34><stop>

Note that the address of 80 is doubled to $A0, and the write bit, 0, inserted as the least significant bit.

To read back from locations $60..63, the following is sent:

<start><$A0><$60><start><$A1><??><??><??><??><stop>

After the second start condition the device address is sent out again, but this time with a read bit, 1, inserted as the LSB, so $A1 is sent: at this point the 8583 starts outputting data until the stop condition occurs.




View of Minerva I²C connections, solder side

## System Extensions

Minerva has a set of System Extensions of its own which serve a range of housekeeping purposes, some of which may be freely manipulated by the user, others which are definitely only for the experienced hacker.

They can be found between the top of the channel table and the base of the common heap. We cannot conceive of anybody who is shortsighted enough to expect the heap to follow on exactly from the end of the channel table, but we've been proved wrong before!

The base of these extensions is pointed to by the System Variable SV_CHTOP at offset $7C (decimal 124).

Thus from SuperBASIC:

```
sx_base=PEEK_L(VER$(-2)+124)
```

The scope of the extensions is quite wide; offering everything from the trivial effects of changing the fonts which all subsequent newly opened channels or jobs use, changing the cursor colour, shape and flash rate through to implementing new keyboard drivers (see the keyboard _asm files on the supplemental disk for examples) to the real heavyweight stuff of changing the device driver linkage pointers and the close routine for the MM.RECHP driver. Play with these latter ones at your own risk!

```
sx_case     equ $00     L       non-zero = user routine on  CTRL-
                                ALT-SHIFT-ENTER (C/A/S/E?)
sx_itran    equ $04     L       input translation routine
sx_otran    equ $08     L       output translation routine
sx_driv     equ $0C     L       MM_RECHP's memory management
                                driver, close entry point
sx_kbenc    equ $10     L       keyboard encoder routine
sx_ipcom    equ $14     L       routine to front end MT.IPCOM calls
*spare*     equ $18     L       RESERVED (routine/table)
*spare*     equ $1C     L       RESERVED (routine/table)
sx_trn      equ $20     L       default i/o translation table address
sx_msg      equ $24     L       default message table address
sx_f0       equ $28     L       default primary font
sx_f1       equ $2C     L       default secondary font
sx_dspm     equ $30     B       real display mode settings (dual
                                screen)
sx_toe      equ $31     B       Turn off enhancements
sx_event    equ $32     B       Keyboard events
sx_fstat    equ $33     B       cursor flash rate, size and color
                                RRRRSCCC
sx_kbste    equ $34     B*12    special key remap table
sx_qdos     equ $40     B*4     returned by MT.INF, VER$(-2)
sx_basic    equ $44     W+B*4   returned by VER$, VER$(0)
*spare*     equ $4A     W*2     RESERVED
sx_ipcrtn   equ $4E     B*2     IPC return codes (experimental)

* Initial RAM based linkages *
            equ $50     L*2     00000000
            equ $58     L*6     BASE+$60
                                00000000 IO_SCAN *
            equ $70     L*4     BASE+$80 OD_SERIO OD_SEROP
                                OD_SERCL
            equ $80     L*4     BASE+$90 IO_SERQ OD_PIPOP
                                OD_PIPCL
sx_con      equ $90     L*4     BASE+$A0 OD_CONIO OD_CONOP
                                OD_CONCL
            equ $A0     L*4     00000000 OD_NETIO OD_NETOP
                                OD_NETCL
            equ $B0     L*8     00000000 DD_MDVIO DD_MDVOP
                                DD_MDVCL MD_SLAVE 0 0
                                MD_FORMT
            equ $D0     L,W,C*3 MD_END 3 'MDV'


*spare*     equ $D9     B*7     RESERVED
            equ $E0             end of system extension
```

Here is a brief summary of the extensions and their use:

sx_case

If this is non-zero it is assumed to be pointing to a user routine which will be called when CTRL-ALT-SHIFT-ENTER is pressed. It is called within the keyboard interrupt routine in Supervisor mode and therefore you should save and restore all registers and do not attempt to allocate or release memory. The top bit should be set for the routine to be recognised.

sx_itran
sx_otran

The input translation and output translation routines are used to translate characters passing through the serial queues as explained in the section dealing with the serial driver in the CONCEPTS section.

sx_driv

The SX_DRIV pointer is the close entry point for the MM.RECHP routine. Although it is unlikely you will use this pointer (it is primarily there for the system's own use to avoid having absolute pointers within the ROM itself) it has the interesting possibility of being used to front-end the MM.RECHP call so that you could monitor some of the memory usage within the machine. Should a subsequent memory violation occur which may be picked up by QPAC's SysMon you might get more information on who did the dirty on your machine! It must however finish by calling the original close routine or you'll be in big trouble. When in doubt leave it alone!

sx_kbenc

SX_KBENC allows a keyboard encoder routine to be added; the best example of how to use this is to examine the _asm files on the supplemental disk for the foreign language keyboard drivers. This is also relevant to SX_IPCOM which front-ends calls to the IPC via MT.IPCOM and is used to manage the ABC keyboard hardware to modify the KEYROW call.

sx_trn
sx_msg

SX_TRN and SX_MSG do a similar thing to the TRA command which has pointers in the System Variables area although the operate when the use default command is issued. Thus altering these pointers will only take effect when a revert to default TRA is issued, this differs from the SV versions which take immediate effect.

sx_f0/f1

SX_F0 and SX_F1 are a pair of pointers that can easily be used by the relative novice. When new screen channels are opened these font pointers are picked up and used. If it is required that only one channel get use of the new font then read the pointer, substitute the new value, open the channel(s) required then restore the previous value. That font will remain attached to that channel until it is closed or in the case of a job, until the job is removed from the TPA.

An example in SuperBASIC:

```
old_font=PEEK_L(sx_base+40)
font_addr=RESPR(font_size)
LBYTES flp1_font_name,font_addr
POKE_L sx_base+40,font_addr
OPEN #3;scr:PRINT #3;'Hello World!':CLOSE #3
POKE_L sx_base+40,old_font
```

sx_dspm

This gives the return byte as described in the MT.DMODE section. It is re-arranged to take into account which default screen you are in. It is not recommended that you change the mode by poking this byte but use the proper MT.DMODE calls − please do as we suggest!

sx_event

SX_EVENT is written to by the keyboard interrupt routines and read (and cleared in some cases) by the scheduler to implement CTRL-SPACE (BREAK) and CTRL-ALT-SPACE (BREAK MultiBASICs). This is safer to the system's health as the original CTRL-SPACE (BREAK) used to try directly to set a flag in SuperBASIC, whilst it could in fact be *moving.* At best this might mean you didn't actually get a BREAK and at worst you could crash the machine by poking the wrong byte into SuperBASIC.

Under the new scheme, the scheduler flags SuperBASIC that a BREAK event has occurred and as the scheduler is running, SuperBASIC can't be moving because nothing else is running! As a secondary precaution, SuperBASIC now invokes supervisor mode whilst in motion so the scheduler can't be called when SuperBASIC actually does move.

If you want to break SuperBASIC, you can do it by setting bit 4: to break all running MultiBASICs set bit 5.

Bits 0 to 3 are reserved for future user events as yet undefined.

| Bit | Key | Event |
|---|---|---|
| 0-3 | Reserved | |
| 4 | CTRL-SPACE | (BREAK) |
| 5 | CTRL-ALT-SPACE | (MultiBASIC BREAK) |
| 6 | CTRL-SHIFT-SPACE | (unused) |
| 7 | CTRL-ALT-SHIFT-SPACE | (unused) |

sx_fstat

You can now set the cursor flash rate, shape and colour by setting the appropriate bits in sx_fstat. The top four bits determine the flash rate. The size bit, if set, produces an underline cursor: if clear it produces the standard QL block cursor. The bottom three bits determine the cursor colour as per the standard 0-7 colour designation. We quite like 76 here!

sx_qdos
sx_basic

sx_qdos and sx_basic allow you to alter the version numbers as returned by calls to MT.INF and VER$. This was put in to allow certain software to be fooled into running on  ; we haven't come across many of these, the only one we know of at present is the German QUILL which expects the MT.INF version of QDOS to be returned as 1G10 instead of 1.10 − patching the '.' in the sx_qdos bytes will allow this program to run on .

For an experimental time, versions 1.86 and maybe above will be using the two bytes at $4E and $4F in the extension area to record some additional status data that comes in from the IPC, but has never been seen before.

The first of these is set whenever serial data bytes are handed on from the IPC to the main processor.

The low order 6 bits contain the serial byte count, which should always be between 1 and 23, so far as we know.

The upper two bits are more interesting, and seem to be flagging serial overrun on ser1 and ser2. The second byte has in bits 6, 5 and 2 the status bits from the IPC that have never been used for anything (it's been rotated and munged a bit from the original input, where these bits were 3, 2 and 7 respectively).

Even with the benefit of a complete dis-assembly of the IPC code, we haven't worked out exactly what they all do, yet, but one is a flag that you have changed the combination of CTRL/ALT/SHIFT keys you're holding down, while holding yet another key... usefulness... zero? More detail may follow, if it's of interest.

## RAM-based linkage pointers

These linkages are pointers to the default list of device drivers. Previously these were absolute pointers within the ROM, but it became policy for these to be pulled out into RAM to permit modification. A full explanation is beyond the scope of this manual but it now becomes possible to replace a QL device driver with your own, without making a copy of it and then ensuring your version slots itself in at the head of the list.}

# Index



## A



## B



## C



## D

# E



# F



# G



# H



# I

# J



# K



# L



# M



# N



# O



# P

# Q



# R



# S

# T



# U



# V



# W



# X



# Z