(Preliminary Version, 1st Revision - Sept. 1983)

Upgrading Dragon 32s to 64k of RAM
by R.W.Hall, National Dragon Users Group.

Disclaimer. Although every care has been taken in the preparation of these
notes, and the technical data have been checked, neither the author nor the User
Group accept any liability for inaccuracies or consequential loss or damage,
however caused.

1. Introduction.

These notes are provided to help the adventurous upgrade their Dragon 32s
from 32k of random access memory (RAM) to 64k, and to give them a feel for
possible uses for the additional memory. The technical detail given is adequate
rather than exhaustive, and assumes that you have some competence in
electronics, and in the programming, an acquaintance with the hardware of the
machine, such as may be found by reading say 'Inside the Dragon' (Smead &
Somerville, published by Addison-Wesley, 1983). The notes are based on articles
I wrote in Dragon Update, and the feedback that I received from other members of
the User Group.

2. Is upgrading worthwhile?

To answer this question let's compare the spec of the Dragon 64 with that of
a 32:

- The Dragon 64 currently retails for £195; it has twice as much RAM as the 32, an
  RS232 interface, and two BASIC ROMs - the additional ROM is used when the 64 is
  in '48k' mode. A Dragon 32 by contrast retails for £50 (if you can find one),
  and secondhand fetches about £25-35. Changing the memory chips will cost you
  well under £15 if you do it yourself (£20-60 if someone else does!), and adding
  an RS232 interface - following the circuit in July 85 'Electronics & Computing'
  - would cost probably another £15. Without the additional ROM however, the
  additional RAM cannot easily be used by BASIC. So, if you don't have a disk
  system, and always write in BASIC, upgrading to 64k is probably not going to be
  useful to you - if however you know at least a smattering of machine-code or use
  a language like FORTH, then even if you haven't got a disk yet, you'll probably
  find the extra memory useful. If of course you have got a disk drive, you'll be
  itching to make the mod.(?!!)

3. First Steps.

The circuit-boards in D32s come in half-a-dozen different varieties, but
there are two basic types - those having 8 32k (by bit) RAM chips, and those
having 16 16k RAM chips. The former are rather easier to modify than the latter
(more common) types. To find out which sort you've got, type the following,
immediately after switching on:-

```
?FEEKH(8FF22) AND 4
```

If the answer is 0 then you've got an 8 by 32k system, if its 4 then your memory
is 16 times 16k chips - if you get any other answer, steady your trembling
fingers and try again!

(The significance of this test bears explaining:-
when the Dragon is switched on, or when reset is pressed, a routine in ROM at
(hex) B3B4 is entered, which sets up the default settings of the two Peripheral
Interface Adapters (PIAs), the Video Display Generator (VGA) and the Synchronous
Address Multiplexer (SAM); until the SAM has been set up correctly, the RAM
cannot be used. To set up the SAM however, the ROM routine on a D32 must know
the memory type, and the 2nd bit at FF22 is programmed as input and used for
this purpose. The D64 reset routine does not need to know the memory type, and
the bit is instead used as an output signal to switch between the two BASIC
ROMs).

If you find that you have (8 times 32k) memory it is very likely that the chips
are in fact 'half-good' 64k units (they are often marked as OKI 3732a).

PAGE 1
It's fairly straightforward to check if you do have these sorts of chips, by trying to write information to the uppermost 16k of the 64k (ie 0C00-0FFF), when there is nothing plugged in the cartridge port, (and then trying to recover it), using the routine given in Appendix 5. However, you won't be able to test all of the upper 32k (if it exists) until you've made the mod to the decoder circuitry. This is, I suggest, the next step to make, in all cases.

4. Modifying the Decoder Circuitry.

The modification to the decoder circuitry is straightforward, and can be added either externally or internally - however, it is only worth making the mod externally (via a board in the expansion port) if you find you have 'half-good' 64k RAMs aboard, you have no diskdrive or intention to get one, and are happy to discover and work with the imperfections in the memory. The internal mod uses a spare gate on one of the 74LS02s - this is the same gate as some people have used to add a 'reverse-video' (green-on-black instead of black-on-green) capability (see Update No. 5) - if you've added this you'll have to first remove it and move it to another spare gate, wired as an inverter.

The details of both the internal & external mods are given in Figs 1 & 2. Both perform a similar function - they stop the 74LS138 (a '3-to-8 decoder') from being activated when write-to-memory is true, and thus prevent the basic 6 cartridge ROMs being spuriously activated when writing to RAM in the memory map above 32k. This circuit is functionally equivalent to that used by the D64 and has the same side-effect, namely:

- If you have any device in the cartridge port between 8000 and FEFF which is activated by the 'CART SEL' signal (pin 32), it will not be possible to WRITE to this device: this is because 'CART SEL' is derived from the LS138 decoder. If the device in an I/O device, then the correct solution is to move its address to FF00 or above using F2 (pin 36) and address lines as the select-device signals.

If it's static RAM, you could separately decode the upper address lines, but do you need it now anyway? (If so, see Appendix D for a possible fix).

The external mod could be built up on any of the prototyping boards which are available from Tandy, Conspusean or Mindrush. However, these are fairly expensive, certainly compared with the two logic gates which make up the circuit; a cheaper option may be to use a 44 way plug and extender board for the spectrum expansion port, and cut it down to 40 ways to fit the Dragon port.

It is possible to make the internal mod, without removing the RAM and CPU, provided you take the usual precautions against high voltages & static (earth the iron, the board and you, and wear no nylon clothes). In detail the mod is as follows:

1) Check that IC26 is a 6821 (PIA), IC33 is a 74LS138, and IC31 a 74LS02 with the gate on pins 1 & 3 unused.
2) Now join pin 12 of IC31 to pin 3 of IC33, and pin 2 of IC31 to pin 21 of IC26 (this is just a convenient W/H line) (IC35, also a 6821 PIA, may be more convenient than IC26 on some boards). Reassemble, and check if it still works.
3) Then b) Join pin 1 of IC31 to pin 5 of IC33; and finally, cut the connection between pin 5 of IC33 and pin 8 of IC33 (this is easier than it sounds - however the hardened steel tips of a good pair of miniature side cutters will do the trick).
4) Test out the circuitry - the effect on normal operation should be nil.

If you have 'half-good' 64k RAMs aboard, you can now begin exploring their potential.

5. Changing the RAM chips.

a) Preliminaries.

The first problem is to get hold of 8 64k RAM chips at a reasonable price. Chips with an access time faster than 300ns will probably be good enough, and the only real subtlety is to get hold of 128-cycle refresh chips, not 256-cycle refresh chips - this excludes in particular the Texas 4164 (but not the Toshiba or NEC or Motorola 4164s!). The 6883 cannot properly refresh 256-cycle chips - although these may be adequately refreshed, miles out of spec., by the action of the VCM taking information from them. I give a possible list of suitable chips.
and their suppliers in Appendix C - at the time of writing it should be possible
to purchase suitable chips at well below £1-50 each. Even at their present low
prices however, I suggest you socket the chips, since this makes fault tracing
much easier - suitable low-profile 16-pin DIL sockets cost about 15p each.

Before beginning the removal and replacement of the RAM chips I would
suggest you remove the SAM and CPU (which are socketed) - removal should be
affected gently, using eg a thin-bladed knife to lever up the chip with as well
distributed a force as possible (again, with everything earthed!). Wrap these
safely away (in tinfoil or on a conducting mat for maximum safety), before
turning attention to the main circuit board (~ make the decoder mod. now, if you
have not already done so).

There are two common designs of mainboard in 332s - these are:
1) the 'Issue 2' board, which comes in both (8 x 32k) and (16 x 16k) versions,
   but with only 8 chips directly on the mainboard,
2) the 'Mark II' board which has room for all 16 (16k) RAM chips on the
   mainboard.

You may well come across others, but the notes which follow should help you
through these too.

b) (8 x 32k) Machines.

Machines with (8 x 32k) memory chips are usually built on 'Issue 2'
mainboards, and the procedure for modifying them is straightforward - you must
remove the 8 old memory chips (usually numbered 1C1-6), and replace them by
sockets, check out all lines and voltages, and install the new RAM chips. The
way you do the removing is largely dependent on personal skill and tools
available. I have used a chip extractor together with an IC desoldering
attachment on a moderate-wattage iron - others swear by desoldering braid, or a
solder extractor, whilst still others cut their losses and clip out the chips.
In all cases take care not to use excessive force, because the lines are plated-
through, and breaking the plating is a sure cause of later malfunction. Also be
warned that it is very easy to bridge lines with solder splashes, especially
just under the ICs, where the lines are not lacquered over and are very close
together. It is for this reason that I would embark on a systematic check that
all 8 address lines (20-7) and control lines CAS, RAS and WE between SAM and RAM
are independent, and that the 8 data lines in and out on each chip are also OK
(see Appendix A, Fig e). I think this test is most safely made with volts on
the board, using a low-voltage high-impedance ohm range on a multimeter (NB -
on some meters the highest impedance ranges use dangerously large battery
voltages). (It is wise to proceed this resistance check with a voltage check,
but note that lines with logic connected to them (2A, BOUT, RAS and usually 27)
give non-zero voltages). If bridges are found, it is often possible to clear
them, if you've bought the right design of socket([]), by prising the plastic
surround away from the soldered-in pins, to inspect the state of the lines
beneath.

c) (16 x 16k) machines.

On these machines the 16 RAM chips are arranged electrically in two banks
of 8, which are activated by different 'Row Address Strobe' signals, RAS0 &
RAS1; the chips are usually 64116S, ( or equivalent, eg 93652906), and instead
of having a single 5V supply (on pin 8), have -5v on pin 1, 5v on pin 9 and 12v
on pin 8 (~check these), these voltages (especially the 12v!) now need to be
rearranged (see the planout in Appendix C) for the 64k chips, which go in bank 0
(1Cl-8) only. In addition the RAS1 signal needs bringing over from bank 1 (pins
4) to become the 8th address line (A7) on bank 0 (pins 9), and the SAM
initialisation routine needs informing of this change of duty by a change of the
signal on pin 12 of the PIA IC15 from high to low.

Owners of 'Issue 2' boards will find they can effect most of this by just
changing over jumpers - for owners of 'Mark II' machines the path is slightly
more thorny, but not as difficult as at first appears.
1) 'Issue 2' boards.

A lot of E2's of 82-83 vintage were built on 'Issue 2' mainboards. These had room for only 8 chips on the mainboard, and many of them saw 86K 3270A were fitted (see above). However, they are even more common in their (16 x 16k) versions, when the 8 extra memory chips were added in sockets above the others; or else a small subsidiary 'patch' board was added, containing 16 RAM chips and the 74LS244 memory buffer - 6 RAM chips remained on the mainboard, with connections to the patchboard taking up the space normally occupied by 1C1, 1C8 (RAMs) and 1C16 (74LS244).

To facilitate their use as either (8 x 32k) or (16 x 16k) boards, 5 jumpers were fitted, just in front and to the right of the power supply connector (looking from keyboard!). The positions of these jumpers are numbered on the board from 1 to 10 (odd positions are at back), and they perform the following functions (which, if you are in any doubt, you should verify!):-

a) The first (leftmost) jumper controls the inclusion, or not, of an inverter in the 1A51 signal line. In position 1 there is no inverter, in position 2 there is.

b) The second jumper controls the voltage on pins 1 of the RAMs. In position 3 it is 5V, in position 4 it is 0V.

c) The third controls the voltage on RAM pins 8. In position 5 it is 12V, in position 6 it is 5V.

d) The fourth controls the use of RAM pins 9. In position 7 there is +5V on the pins, in position 8 it is 1A51/27.

e) The fifth (rightmost) jumper controls the voltage on pin 12 of IC35. In position 9 this is 5V, and in position 10 0V.

In (16 x 16k) mode the jumpers will be set to positions 1, 3, 5, 7, 9, and there will be a flying lead from position 8 taking the 1A51 signal to the patch-board.

The straightforward way to modify these boards is as follows:-

a) Remove the patch board and its connecting lines, saving the 74LS244.

b) Remove IC2-7, C11, C59 and any other capacitors on the line connecting pins 9 of IC1-8 to 0V on the RASO line.

c) Clean up the board, and fit 16 pin sockets in positions IC1-8, and a 20-pin socket in IC16.

d) Set the jumpers to 1, 4, 5, 6, 10.

e) After checking all voltages and lines, install the 8 new memory chips in IC1-8 and the 74LS244 in IC16, and step back.

This method of modification produces a nice clean job, with everything easily accessible in case of trouble. However the patchboard may well have its chips installed in sockets, and it is then tempting to try installing the new memory chips on this patchboard. I suggest the following route, which is for an issue 1 patchboard, with the proviso that I gave up on it!:

a) Remove IC1-10 etc from mainboard as described in b) above.

b) Remove IC1-10, R1 & C23 from patchboard.

c) Take the flying lead, labelled 'RAS1', from jumper position 8 and connect it to the wire going to pin 4 of IC1 on the patchboard (le to pick up RASO) - it's probably also worth linking the junction of R1 & C23 on the patchboard to the junction of C59 & R66 on the mainboard, to produce the same effect.

d) Now for the catch - unfortunately, on the issue 1 patchboard, the 5V supply to pin 20 of IC11 (the 74LS244) is chained in with the lines connecting pins 9 on IC1-10. This chain must be broken, and IC9 & 10 relinked to IC1-8. I suggest breaking the line, which runs underneath the board, under IC3 (at back edge of board, near cent-re); and at the front right hand edge by IC10. After reinstating a connection between pin 9 of IC10 (by deft unsealable soldering!) and the wire to pin 9 of IC1, the difficulties are (I think) over.

e) Set jumpers to 1, 4, 5, 6, 10; check out.

f) Put new memory chips in patchboard sockets IC2-7, IC9 & 10, and return 74LS244 to IC11 if you removed it....now turn on, and the best of luck.
ii) 'Mark II' boards.
The sixteen memory chips are all installed on the mainboard and the two banks are usually numbered ICI-8 (bank 0) and IC24-43 (bank 1). Both sets of chips are to be removed, but only ICI-8 are replaced with sockets. (In physically, the chips are arranged as a group of 9 and a group of 7 - IC43 is next to IC8). Because there are no jumpers to help you on these boards all the changes must be made by cutting/rerouting lines. These wiring changes are as follows:

1) the -5v line on pins 1 should be chopped (if necessary at both ends of the bank) and left to float; I odd this modification on my board, a Mark II, Issue 6, by cutting through the line on the underside of the board, near the FPU socket, but just after the connection to TBA and B69. On most boards the -5v line is not a through-line, but in cases where it is the connection between the two ends of the chain must be reinstated with a patch wire. (Note. The pins 1 are not connected internally on most 48k RAMs - it is not now to leave the -5v line connected however, since on some chips (eg Motorola 6665) this pin is pulled up internally to +5v!; moreover if -5v is acidentially cross-connected to any of the RAM, SAM or CPU chips it will blow them!)

2) the +12v line (on pins 8) must be similarly chopped - it in general is a through-line (as route to the cartridge port) and so the through-circuit must be completed by a tiny patch-wire.

3) the +5v line (pins 9) is then chopped at both ends of the bank, and the voltage diverted via the old 12v line connecting pins 8. I suggest you sever the line between CI2 & CI1, and again between IC5 & 43/C50.

4) the old 5v line connecting pins 9 now becomes the eighth address line. Remove any bypass capacitors on the line, then take a patch-wire from the line to that connecting pins 4 of bank 1. Using the old 5v line in this way as a signal line probably loads up the signal with rather more capacitance (80pF?) than is present on the other address lines (because the land is quite wide) but this does not seem to have any adverse affect on the circuitry - note that connecting the signal across in the way suggested ensures the inclusion of the 22 ohm damping resistor (R12) between SAM and RAM.

v) when all this is done, it remains to change the signal on bit 2 of FF22.

This is done by disconnecting pin 12 of IC35 (6621, PALA) from B73 (a 'pull-up' resistor connected to 5v), and wiring it to ground.

This ends the mods - a subsequent careful test of all voltages on the RAM, SAM & CPU sockets, followed by a systematic test for bridges or other errors, prior to replacing the chips, is particularly recommended (see remarks in 5b). Testing out by software in the event of malfunction is largely fruitless, because most faults lead to totally dumb systems - however, provided you have checked that the voltage rails are correct (and in particular that the -5v and 12v lines have been routed away from the RAM), all changes should be reversible. (Note that failing to perform step v) (or equivalently on an Issue 2 board, failing to change jumper 5 from posn 9 to posn 10 ) will produce a running system but with only 16k of memory visible to RAMQC.

In the cut-throat world of micro-manufacturing, Dragon Data seemed to have shown commendable ingenuity to use what components they could get hold of, especially RAM chips. The result is that, although the three boards described above are the most common, there exist a number of less common varieties, of which I don't have 'hands-on' experience. One of these is the so-called 'Siemens' board, built using 8 Siemens HY8-4232s. Though numbered as if 32k chips, these were in fact double 16k units. They were 16-pin chips but were installed on Issue 2 boards, and the instructions given above should suffice to see you through.

Secondly, there are later versions of the 'patchboard', I believe, and on some of these D8 actually installed 32k chips! A careful inspection of your board (if it's not an Issue 1, dealt with above, and if it doesn't already contain only 8 x 32k chips, when the answer is clear!), should tell you if its
worth keeping in place or not.

In all cases of doubt, make sure you arrive at the connections between the principle components outlined in Appendix A.

6. What can be done with it all.

Let's now turn to ways of using the new memory which do not require a disk-drive.

When the Dragon is first switched on, it is in Memory-Map Mode 0, and the new 32k of RAM is tucked away out of sight. To access the new memory we must either switch to Map Mode 1, or switch from Page 0 to Page 1 in Map Mode 0. In Map Mode 1 the new memory occupies 0000 to FFFF and we have no access to the BASIC & cartridge ROM's: neither can we access the last 256 bytes of RAM (FFFF-FFFF) which are reserved for I/O. In Map Mode 0, Page 1, the ROM automatically adds 32k to the cpu (but not VIC) addresses before sending them to RAM so that the new memory appears to lie at 0-FFFF (in place of the true lower memory, which is now inactive-instead). In all those modes however (Map Mode 0, Page 0; Map Mode 0, Page 1; Map Mode 1), the VIC has access to all 64k of RAM (and never to the ROM), which it sees in the 'correct' order.

It's fairly straightforward to get from Map Mode 0 to Map Mode 1, and vice versa, in machine code or FORTH (in fact any language which doesn't need the ROM routines for a bit), by toggling the addresses FFFD & FFEE (see Appendix A) - but you must turn the interrupts off first. An example of this is given in Appendix B, program 1, which is a routine to copy blocks of data between low & high RAM, or between ROM & high RAM. It may be used to copy the BASIC & cartridge ROM's into high RAM, and then run them - it is then possible to customise the BASIC/DOS and also have easy access to the memory from 0000 to FFFF.

Program 2 uses a high-res text generator (NOT provided!) resident in high-memory to write text from BASIC on to a high-res screen, also in high-memory. (This routine thus effectively gives BASIC routines use of 8k of graphics RAM for nothing - note that it makes use of the fact that the VIC will continue to display from high-memory whilst the processor is in Map Mode 0.)

Access to and from Map Mode 0, Page 1, requires care, because the system stack has to be reset each time; page-switching is achieved by toggling addresses FFDB or FFDE, again with the interrupts disabled. In program 3 the two pages are used to give two wholly independent areas for DOS, so that two independent 32k programs can be run. There is no reason why the two programs need be wholly independent however, once the essential features of the changeover are understood.

In all these investigations of the Dragon's Memory Modes, the Motorola data sheet for the N56883 (reproduced in 'Inside the Dragon') makes an invaluable, if concise, reference.

7. Flex versus OS9.

Many people contemplating upgrading to 64k have an eye to running either OS9 or Flex (OS9 is not available for Freider 'Delta' disk systems).

Both Compuverse Flex and Dragon Data OS9 operating systems will work with an upgraded machine (provided you've done the decoder mod.) - however, whilst Flex will boot up straightforward, a slightly more roundabout method is needed for OS9, because the absence of an RS232 port at FF04-5 causes the bootstrap routine to hang up. To get round this, proceed as follows:-

1. Type 'BOOT' - the words 'OS9 BOOT' should appear on the screen, and a few seconds later all disc activity ceases, but with the disk still selected and turning.
2. Type 'RESTART'.
3. Type 'FORQ LHFPO3.bib5; EXE 9735'.

Success should then be yours.

A brief comparison of the advantages of the two systems may be useful.

Flex is structurally simpler than OS9, being essentially a single-user, single-tasking operating system of mid 70's vintage; it has been described by Mike James (in 'The OS9 Companion') as 'almost without argument the standard operating system'.

PAGE 6
The system for the 68000 and certainly the best single-user operating system for the 6809. By contrast, 689 is rather complex, being multi-user and multi-tasking; the multi-user facility is largely academic for Dragon users, and the main use of multi-tasking is to run a 'background job' to the printer whilst proceeding with something else on the screen - there just isn't enough memory to do much else anyway. (‘Printer-spooling’ is a feature of some implementations of Flex, but Composense decided not to implement it on the Dragon for rather purist reasons). The Flex system comes with a 21 by 24 character-set, with a facility for 'windows'; the 51 by 24 character-set is optional in 689 (type '6G51' to get it) but when it is installed, there is only about 40k or 50k free, as against 48k for Flex. Where 689 scores is on the availability of cheap(ish) software - even before the recent price cuts by Touchmaster and Cotswold Computers the 689 packages were more attractively priced that their Flex counterparts, and with the recent dramatic price reductions this is now even more true. (This is rather paradoxical, since one of the claimed advantages for Flex is the wealth of software available for it - there is, from specialist firms like Mindrush, if you'll pay £3000 a time!). The only really available software for Flex is the editor/assembler, DBASIC, 689 (also available under 689), and a wordprocessor. The assembler is quite nice (it allows 'macro's'), but not perfect (eg. it doesn't optimise the code for PC-relative forward references ); the editor is an old-fashioned line-editor, as is the the 689 one; DBASIC is essentially a coping-over of the Microsoft RAW into RAW - it has its uses but has none of the speed or aids to structured-programming of 689's BASIC09. (And I haven't tried the Flex word-processor, but this is written using 689's Stylegraph). Overall, I feel that the Flex/Editor/Assembler is a very powerful package for developing machine-code programs, but 689 has more to offer in other lines (eg C & Pascal), if you can live with its memory limitations. At the end of the day it's the old tale of horses for courses - why not play safe and buy both? 8.

The Future - 256k?

256k RAM chips are now the same price 64k chips were 18 months ago (£3-50 or so). These chips have the same pinout as the 64k ones, except that pin 1 is now used for a 9th multiplexed address line. A recent article in Byte (September 85, pages 247 to 254) showed how such chips could be incorporated on the Atari 800XL, and how the refresh problem (all 256k chips are 256-cycle refresh) could be overcome; it is likely that a similar method could be applied to the Dragon (or 'CAS before RAM' refresh could be implemented). The outstanding problem is how to manage the memory map (the Atari already has a dedicated register for this), and who will produce software to run on a 256k machine. A suitable 'Rolls-Royce' quality memory management unit for the 6809 does already exist - it's called the Motorola 6829 and costs about £25. If you're prepared to install one (NOT trivial!) you could in principle have a machine capable of running 689 Level Two (If you can persuade someone to sell this). Ordinary mortals however, would be advised to wait awhile, until Nesper Composense & Eurohard sort out their own upgrade path. If you socketed your chips, you can wait safe in the knowledge of being prepared for the next leap forward.

9. And Finally...

I'd like to acknowledge helpful feedback from Paul Grade, John Russell, Alan Butler, Arne Eriksson, and John Payne. I hope that you find these notes useful - if you feel you've got a problem however, or if you just want to talk some aspect of this upgrading through, write to me via Paul, or phone me on 4227x 2999 evenings). Your comments will be very welcome - in keeping the Dragon a viable machine, we're all in this together!
APPENDICES.

Appendix A. Memory Control on The Dragon.

The MS6841/74LS743 Synchronous Address Multiplexer or SAM plays the central role in the architecture of the Dragon, interfacing the control of the RAM, ROM, 6809E cpu, and VDG, and providing the E and Q clock signals. All this is accomplished in a standard IC package of only 40 pins, and the penalty for this is that, having no connection to the cpu data bus, the RAM must receive its orders from the cpu on the address bus. 32 addresses, from FEOU to FFDU, are used for this purpose, and control 16 bits of information; writing (anything!) to the odd addresses sets a bit, whilst writing to the even addresses clears it.

Most well-known of these operations is the so-called 'speed-up poke' (POKELAPOKE,0 and its reverse, POKEALPFDE,0); however, of more concern to us here is the Flag Type Bit (CT), set by FFDE, cleared by FFDC; the Page Bit (PI), set by FFDC, cleared by FFDE; and the Memory Size Bits (W & H) set/cleared by FFDU/FFDA & FFDC/FFDC respectively. Of the remaining addresses, FFDD to FFDF control the 2 clockrate bits (H0 & H1); FFDC to FFDB control 7 bits (V0 to V6) giving the base address of the current video-RAM (in units of 512 bytes); and FFDU to FFDC control 3 VDG Mode bits (V0 to V2).

The SAM interleaves requests from the cpu for data, with providing data for the VDG. Depending on the map mode and the memory address sent out by the cpu, the SAM will either pass the address required to RAM, or will activate a ROM or output device, (via signals sent to the LS138 decoder); meanwhile, the address of the next byte in video-RAM for output to the VDG is generated in the SAM itself.

There is more in passing an address to RAM than meets the eye, because to reduce the size and complexity of the RAM chips, the address presented to them is divided in two parts, called the row address and column address; this is the 'address multiplexing' process for which the SAM gets its name. When 32K or 64k chips are in use, the SAM first presents the lowest 8 bits of the address to the RAM chips on its output lines 20 to 27, and asserts the Row Address Strobe (RAS); about 0.1 microseconds later (!) it presents the upper half of the address on the same lines, and asserts the Column Address Strobe (CAS) - about 0.1 to 0.2 microseconds later again, the data output from the RAM chip has stabilized, and is read on the data bus (or, if write-enable (WE) is activated, data is read from the bus into memory). Now when 16K RAM chips are used, only 7 lines (20 to 26) are needed for addressing; 27 is now programmed (via ML) to act as a second Row Address Strobe (RASI), thus allowing the use of two banks of 16k chips.

Finally, the SAM controls the 'refreshing' of the RAM chips. Dynamic RAM chips have a simple structure of one transistor and capacitor per bit (or cell), and the charge leaks away from the capacitor over a few tens of milliseconds. The RAMs are so constructed however that every time a row address is presented to them, all the cells with this row address have their information read out into buffers - later in the cycle this information (or new information if there is a write-to-memory) is read back into the capacitors, so that the whole row is refreshed at once. The SAM inserts a series of these 'RAM-only refresh cycles' from time-to-time to keep all the cells updated. As a final subtlety, not all 256 RAS combinations need to be addressed on most chips, only 128. However some chips (notably the Texas 4164) do require 256 refresh addresses, and cannot be refreshed by the 6883d.

Layout of the Dragon.

Both Dragon & Tandy machines closely follow the recommended circuit given by Motorola in the M6809 data sheet. On all the boards the SAM output lines (20 to 27, RASD, RASD/27, CAS & WE) are linked to the corresponding pulls on the RAMs (AO-A6,RASJ & A7, CAS & WE) by low-value damping resistors; on the 'issue 2 boards however, there is optionally also an inverter between 27 and A7, presumably to allow the good upper half of 'half-good' 64k RAMs to be used where appropriate; it can safely be left in circuit when upgrading. As for the data lines on the chips, the inputs Dn are connected directly to the data bus from

PAGE 8
the CPU, whilst the outputs from (marked as 'O' on most RAM datasheets) go via a 74LS244 buffer IC to the data bus, and via a 74LS273 latch to the VDU; data is strobed into the latch under control of RAS0, whilst the 74LS244 memory buffer is activated by a signal from the 74LS138 decoder (which in turn decodes signals 20, 31, 32 from the RAM — see block diagram, Fig 6). Note that, on MKII & Issue 2 mainboards, CL1 is connected to data line D7, and so on in reverse order till IC8, which is connected to D0.

Appendix B: Some programming examples.

The three programs given in Listings L1 to L3 have already been briefly introduced in the main text. At the heart of each are a series of short machine-code routines; these are coded in hexadecimal and placed in BASIC DATA statements, (one instruction per statement). The surrounding BASIC program loads the machine code, by default into the cassette buffer at (hex) 200, and then wherever necessary modifies it, or installs jumps in the appropriate BASIC input/output hooks. As a rule these m/c routines perform their manipulations with the interrupts turned off; this is because in normal operation the interrupts are vectored, by jump instructions in locations (hex) 100 to 112, into routines in ROM; unless both the vectors and service-routines are copied over to RAM the system will crash when an interrupt occurs. The most frequent interrupt comes from the 50Hz clock, which can be turned off separately in BASIC, by executing "POKE65535,46H4".

Program 1: Here the machine code begins by saving all registers and turning interrupts off; it then executes a simple loop in which data is picked up from an address pointed to by A, in one map mode, and deposited in the address given in the × register, in another map mode. The map mode for picking up the data, for depositing it, and the final map mode to return to, are all set up by BASIC, as are the start & finish addresses in the first field, and the start address in the second. By copying from 0000 to E000 (if you have a disk, SIFF if not) in mapmode 0, to 8000 on in mapmode 1, and returning to mapmode 1, you will smoothly transition from running BASIC in ROM to running it in RAM, giving space for machinecode etc from E000 (0000) to PEFF.

Program 2: This uses the Compassense Flex High-res text-generator resident in RAM above 32k to write text from BASIC to a high-res graphics screen at E000 in RAM. The text has true lower-case, and a 5l by 24 format. (A similar arrangement could probably be made with other similar text-generators.)

To use the program first boot Flex (not provided!) and turn off any options set for the terminal (eg unselect 'pause at end of page' by typing 'TTSET, UF=0'), then type 'HOLD' to return to normal BASIC. Finally load and run the program — the BASIC may be deleted after it has run, leaving the m/c routines.

The program works by intercepting the BASIC 'output a character' hook; if the output is heading for the disk, cassette or printer (9D nonzero) then it is sent on its way — the version shown assumes you have DragonBAS, so amend line 40 to 39,12,12 if not. If the output is heading for the screen, then the map mode is changed to 1 and the Flex 'output a character' routine or the 'output a crit' routine entered; (the addresses of these and other useful Flex routines are to be found in the Flex 'Advanced Programmer's Guide'). Finally, it is necessary to keep the VDU looking at the right graphics screen, since there is a BASIC routine called everytime a character is input which resets it to look at the normal text screen at (hex) 400. A simple modification is hooked into the input ram hook to achieve this.

Program 3: This program uses page-switching, to run two independent BASIC or machinecode programs in the two 32k pages of memory. The program is configured to run with DragonBAS, and to remove this feature change line 140 (JMP 49917) to '140 DATA 39,12,12'.

The code comprises 3 sections: SNOPIT, which dups the stack pointer in "HOLD", switches pages, and reloads the SF from the corresponding location in the other page; VDI, which keeps the Video Display looking at the right text-screen (400 in page 0, 8400 in page 1); and finally MOVIT, (a derivative of
program 1), which sets up page 1 initially, by copying page 0 into it and then modifying some locations.

After running the BASIC, type 'Hex' to remove the redundant code, but leave the machine code. Typing or executing 'DECC' will enter SNOPIT, and send you to the other page. SNOPIT could also be linked into the 'End-of-BASIC-line' hook at 19A-19C, or at a pinch into the 50Hz interrupt routine. The routine will work correctly (77) for all BASIC programs which use text only, and m/c routines which use the BASIC I/O. It will also generate graphics correctly, but in page 1 will not display them unless the BASIC statement 'SCREEN a,b' is expanded to 'SCREEN a,b; FORK SHFEJ3,C' (and equivalently for m/c programs).

Use of FORTH. Those wishing to use FORTH will find it easy to change map/page nodes by defining words such as C64, thus:

```
HEX : C6- 0 FFD C ! ;
```

However, you will need to turn interrupts on and off – the following FORTH word, CLOCKOFF, would usually be sufficient.

```
HEX : CLOCKOFF 34 FFD3 C ! ;
```

but a more satisfactory solution is to define a words 'IF and 'ION in machine code as follows:

```
HEX CREATE 10F
1250 , AEAA , 6E94 , SHUDGE
CREATE 10N
1250 , AEA1 , 6E94 , SHUDGE
DECIMAL !S
```

(This screen should work with Oasis/Draka Data & Tele- FORTHS. The m/c AEAA, 6E94 translates as LEX ,F++ JHF (,X) and is the 'in-line' code for the FORTH terminating-word 'NEXT').

Appendix C. Some suitable 64k RAM chips and their suppliers.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Number</th>
<th>Possible Stockists</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAI</td>
<td>85786</td>
<td>Manhattan Skyline</td>
<td>0628-75851</td>
</tr>
<tr>
<td>Hitachi</td>
<td>854864</td>
<td>Farnell</td>
<td>0532-68311</td>
</tr>
<tr>
<td>Motorola</td>
<td>MCM6665</td>
<td>STC</td>
<td>0279-26777</td>
</tr>
<tr>
<td>Kostek</td>
<td>H84264</td>
<td>Coldia</td>
<td>0734-586191</td>
</tr>
<tr>
<td>NEC</td>
<td>805164</td>
<td>Farnell</td>
<td>0532-68311</td>
</tr>
<tr>
<td>Toshiba</td>
<td>TM4164</td>
<td>Vancooper</td>
<td>0703-684555</td>
</tr>
</tbody>
</table>

(All of these have the same pinout (see fig C2), with pin 1 being marked as N/C, except for the MCM6665, for which the pin is connected internally to 5V; also given in the figure (C1) is the pinout of a typical 16k memory chip (HR4116), usually fitted to the Dragon 32.

All the 64k chips above are 126-cycle refresh types (in contrast to the Texas 4164 which is a 256-cycle chip and is not suitable).

Finally, the stockists listed above are (apart from Manhattan Skyline) often not the best sources for small quantities. The following four firms, Technosonic (01-208-1177), Midwinter Computer Co. (0379-898751), Watfords Electronics (0923-37774), Happy Memories (054-422-616), have recently been offering suitable chips at very competitive prices (eg less than £1 a chip for HR4164s at Happy Memories). Check the latest position in 'Wireless World' or 'Electronics & Computing'.

Appendix D. An alternative decoder mod., allowing use of static RAM in the cartridge port.

The decoder modification suggested in the main text is the logical one to use following the Motorola data sheets, and is essentially the same as that employed on the Dragon 64. However, it does have the disadvantage of not easily
allowing static RAM in the cartridge port, because the cartridge-select line is disabled during all write operations. It appears that the following alteration to the internal mod. would overcome this—instead of gating the decoder logic (IC33, 74LS138 & IC31, 74LS02, with R/W (the read/write line from the processor), the signal WE (write-enable) is used. To do this, connect pin 2 of IC31 to the land joining pins 3 of the RAM chips (rather than to pin 21 of IC 26); the other connections, between LS02 and LS138, remain the same.

It must be emphasised that this is a rather speculative mod., using an undocumented property of the WE signal—namely, that WE does not go low when memory above FFFF is addressed in Map Mode 0—this has been determined by experiment only.

For this reason, I do not recommend this mod. Having said this, it would be a great convenience to have static RAM available in this way when using the page-switch facility, since the system stack and a data interchange area could be maintained in it (contrast Program 3 of Appendix B).

BASIC

10 'MOVIT 23-JAN-84
20 DATA 34.7F
30 DATA 1A, 50
40 DATA 8E, 00, 00
50 DATA 10, 8E, 00, 00
60 DATA 87, FF, 00
70 DATA A6, 00
80 DATA 87, FF, 00
90 DATA A7, A0
100 DATA 8C, 00, 00
110 DATA 2F, FI
120 DATA 87, FF, 00
130 DATA 35, FF

ASSEMBLER

PSHS X,Y,A,B,DP,CC,U  Dump the lot
ORCC $50
LDX $0000
LDY $0000
STA  $FF00
LDA X+
STA  $FF00
STA  $FF00
CMPX $0000
BLC L
STA  $FF00
PULS X,Y,A,B,DP,CC,U,PC

All done.

PRINT"movit here";PRINT"GIVE ME A HOLE TO LIVE IN"; INPUT BA
160 IF BA>300 GOTO 170 ELSE BA=512
170 FOR I=BA TO BA+30
180 READ VS; V=VAL("&H"+VS)
190 POKE I, V
200 NEXT I
210 PRINT"TAKE DATA FROM"; INPUT ST, FI
220 PRINT"IN MAP MODE (32K) OR 1(64K)"; INPUT MI
230 PRINT"AND PUT IT AT"; INPUT NA
240 PRINT"IN MAP MODE"; INPUT MO
250 PRINT"RETURN TO 32K(0) OR 64K(1) MODE"; INPUT MR
260 IF MR=0 THEN POKE BA+28, &HDE ELSE POKE BA+28, &HDF
270 IF MI=0 THEN POKE BA+13, &HDE ELSE POKE BA+13, &HDF
280 IF MO=0 THEN POKE BA+18, &HDE ELSE POKE BA+18, &HDF
290 X=INT(ST/256); Y=ST-256X
300 POKE BA+5, X; POKE BA+6, Y
310 X=INT(FI/256); Y=FI-256X
320 POKE BA+22, X; POKE BA+23, Y
330 X=INT(NA/256); Y=NA-256X
340 POKE BA+9, X; POKE BA+10, Y
350 EXEC BA

PROGRAM 1.
**BASIC**

10 'FLEXT 8-MAR-85
20 DATA 0D,6F
30 DATA 27,03
40 DATA 7E,D9,17
50 DATA 32,62
60 DATA 34,7F
70 DATA 1A,50
80 DATA 87,FF,DF
90 DATA B1,0D
100 DATA 27,0A
110 DATA BD,CD,12
120 DATA B7,FF,DE
130 DATA BD,07
140 DATA 35,FF
150 DATA BD,CD,24
160 DATA 20,F4
170 'CHANGE VDG PAGE
180 DATA 34,02
190 DATA B6,F0
200 DATA B7,FF,22
210 DATA B7,FF,C5
220 DATA B7,FF,C3
230 DATA B7,FF,C0
240 DATA B7,FF,D3
250 DATA B7,FF,D1
260 DATA B7,FF,CF
270 DATA B7,FF,CC
280 DATA B7,FF,CA
290 DATA B7,FF,C9
300 DATA B7,FF,C7
310 DATA 35,02
320 CLEAR 500
330 PRINT "Flext here":PRINT "GIVE ME A HOLE TO LIVE IN": INPUT BA
340 IF BA<000 GOTO 360 ELSE BA=512
350 FOR I=BA TO BA+79
360 READ V$:V=VAL("$H"+V$)
370 POKE I,V
380 NEXT I
390 X=BA-Y=INT(X/256):Z=X-256*Y
400 POKE &H189,Z
410 POKE &H189,Y
420 POKE &H189,Y
430 POKE &H167,87E
440 X=BA+35;Y=INT(X/256):Z=X-256*Y
450 POKE &H16C,Z
460 POKE &H168,Y
470 POKE &H16A,87E

**ASSEMBLER**

Start TST $6F Check output device
BEQ SCRN If 0 it's O/P to screen
JMP $D917 If anything else send to DOS hook
SCRN LEAS 2,S Drop return address off stack
PSHS X,Y,A,B,U,DP,CC
ORCC $350 IOF
STA $FFDF Map mode 1
CMPA $5DD CRLF
BEQ CRLF
JSR $CD12 Char O/P
FINISH STA $FFDE Map mode 0
BSR RESET
PULS X,Y,A,B,U,DP,CC,PC
CRLF JSR $CD24
BRA FINISH

RESET PSHS A
LDA $F0 Set VDG mode
STA $FF22 Now set
STA $FFC5 VDG
STA $FFC3 Base page to
STA $FFC0 $6000
STA $FFD1 which is where
STA $FFCF the FLEX
STA $FFCA HIRES
STA $FFC9 textscreen
STA $FFC7 lives
PULS A,PC All done

**PROGRAM 2**

**PAGE 12**
BASIC

10 'HOLE-STACK STORED HERE
20 DATA 0, 0
30 'SWOPIT - SWITCHES PAGES
40 DATA 34,7F
50 DATA 1A, 50
60 DATA10, EF, 8C, F6
70 DATA 32, BC, 75
80 DATA B7, FF, D4
90 DATA 10, EE, BC, EC
100 DATA 35, FF
110 'VDG - CHANGE VDG PAGE
120 DATA 7D, 00, 6F
130 DATA 27, 03
140 DATA 7E, D9, 17
150 DATA 8D, 80, 0C
160 DATA 32, 62
170 DATA 34, 16
180 DATA 6E, FF, C8
190 DATA A7, 0A
200 DATA 7E, A9, 41

ASSEMBLER

10 'HOLE-STACK STORED HERE
20 DATA 0, 0
30 'SWOPIT - SWITCHES PAGES
40 DATA 34,7F
50 DATA 1A, 50
60 DATA10, EF, 8C, F6
70 DATA 32, BC, 75
80 DATA B7, FF, D4
90 DATA 10, EE, BC, EC
100 DATA 35, FF
110 'VDG - CHANGE VDG PAGE
120 DATA 7D, 00, 6F
130 DATA 27, 03
140 DATA 7E, D9, 17
150 DATA 8D, 80, 0C
160 DATA 32, 62
170 DATA 34, 16
180 DATA 6E, FF, C8
190 DATA A7, 0A
200 DATA 7E, A9, 41

PROGRAM

300 DATA 2F, F7
310 DATA 86, D5
320 DATA A7, 8C, C6
330 DATA 4A
340 DATA A7, BD, 7F, 31
350 DATA 86, 0A
360 DATA A7, BC, D6
370 DATA 4C
380 DATA A7, BD, 7F, D1
390 DATA B7, FF, DE
400 DATA 35, FF
410 CLEAR 500
420 PRINT "Swopit here"; PRINT "GIVE ME A HOLE TO LIVE IN"; INPUT BA
430 IF BA=300 THEN BA=812
440 SWOPIT=BA+2; VDG=BA+22; MOVIT=BA+45
450 FOR I=BA TO BA+92
460 'V$=VAL("&H"+V$)
470 NEXT I

500 EXEC SWOPIT
505 FOR I=BA TO BA+92
510 'V$=VAL("&H"+V$)
520 POKE &H167, Z
530 POKE &H168, Y
540 POKE &H167, &H7E
550 EXEC MOVIT
560 EXEC SWOPIT
570 EXEC SWOPIT