Expanding the C128

Increasing System RAM to 256K and Beyond

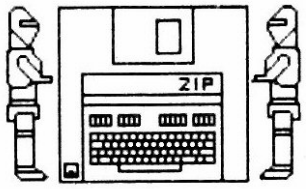
by
Richard Cursio

This material originally appeared in issues #30 and #31 of

Twin Cities 128

Expanding The C128
by Richard Curcio

Expanding the C-128 to 256K
(Part one: Banks 2 and 3)



Editor's note: Before attempting this project you may want to wait until issue number thirty-one for the additional articles to upgrade your C-128 to 512K !

Perhaps my disdain for "memory gluttons" is a hold-over from my days with the Vic-20 and (gasp!) Timex-Sinclair. However, when I saw that the designers of the C128 had allowed for the possibility of that machine having RAM BANKs 2 and 3, even though the Memory Management Unit (MMU) and other hardware make no such provision, I couldn't resist thinking about ways to make that possibility a reality.

I set down certain specifications for this project:

1) No additional I/O registers should be required to select the new banks. Since the operating system already "thinks" it has BANKs 2 and 3, they should be accessible through existing MMU registers using the normal Kernal and Basic commands.

2) The circuitry should be as simple as possible. It would not be worthwhile if a dozen or more ICs (not counting DRAMs) were needed to merely double the amount of memory.

3) Minimal changes should be made to the existing C128 circuitry, including retaining the existing 128K of RAM.

I feel I have met all these specifications. However, because of dimensional constraints in the low-profile 128, item two should be amended to "...as simple as possible ELECTRONICALLY." It is not so simple to make everything fit into the flat case. On the other hand, the 128D has ample room and the wiring of the extra DRAMs in that machine is much simpler.

This article assumes you are familiar with proper soldering techniques and have some knowledge of electronics and expertise in the building and modification of same. If the foregoing does not describe you, it's best to leave this project to someone who does have those qualifications.

The heart of this modification is the installation of a second Commodore 8722 MMU. It is suggested that before beginning this modification, if possible, you test your extra MMU for a few days by removing the old one and plugging the new one into the MMU socket. Removing such large ICs often results in many pins becoming bent. Attempting to straighten damaged pins can cause them to break. I have found a right-angle screwdriver useful in chip removal. The trick is to pry the chip up a little at a time, first at one end then the other. Don't rush it. It is further suggested that you read these instructions thoroughly before you begin. Do not skip information just because it does not apply to your model of the C-128.

A list of the other parts required for this project, and possible sources for the MMU, are given at the end of this article. It is possible to arrange things so that this modification can later be up-graded to 512K.

Bear in mind that presently no available software makes use of the currently non-existent BANKs 2 and 3. Well written programs, such as the Power Assembler (Buddy) can easily access the new banks as it is written. At least for now any other software will have to be modified or written by users who perform this modification. As it will be shown, the new banks can even be utilized in Basic.

Disclaimer

This modification will render any warranties on your equipment null and void. Neither Parsec, Inc. or the Author assume any liability for any reader's or purchaser's implementation of these instructions. All information is believed to be accurate.

Theory

Run this short program

```
10 FOR I= 0 TO 3:BANK I :POKE49152,I :NEXT
20 FOR I= 0 TO 3:BANK I :PRINTPEEK(49152), :NEXT
```

on a stock, un-modified C128 and you will get:

```
2  3  2  3
```

because when it POKEs and PEEKs Banks 2 and 3 it is really accessing Banks 0 and 1. And, since the POKEs to Banks 2 and 3 replace the values in Banks 0 and 1, the values PEEKed from Banks 0 and 1 are the same as 2 and 3.

Once the modification described herein is completed, the result of running the above lines will be:

0 1 2 3

indicating that Banks 2 and 3 are real. They can be accessed with the approriate BANK statement and the VIC can display them by altering bits 6 and 7 of the RAM Configuration Register at $D506.

Refer to figure 1. Above the dotted line is the relevant portion of the C128 circuitry. Pin numbers in parentheses are for the 128D. Below the dotted line is the added circuitry. (The additional DRAMs are not shown.) The original MMU (MMU-1,) generates one of two Column Address Strobes, depending upon the currently selected Bank, the range of memory being addressed, and several other factors, determined by the contents of the MMU's internal registers. The MMU also sends several signals to the PLA, telling it to generate ROM selects, I/O or /CASENB, which selects RAM. The VIC chip provides a "master" /CAS. Note that when the MMU registers at $FF00-$FF04 are being addressed, both /CASs from the MMU remain at logic 1 (inactive). That is how those registers are made to appear across all configurations.

Once the additional circuitry is installed, the original /CASs from MMU-1, combined with /CAS0 from the second MMU (MMU-2,) form a 3-bit "CAS code". This 3-bit code is applied to the A, B, and C inputs of IC Z2, a 74LS138 1-of-8 decoder. We get four 64K RAM blocks because four of the codes are invalid, so four of Z2's outputs are unused. ICs Z3 and Z4 detect and condition a number of signals, and let the two MMUs work together without interfering with each other. Adding a second MMU is much simpler than attempting to reconstruct the required logic with separate ICs.

There are two limitations to this arrangement: None of the additional memory is available in 64 mode, (For this modification, at least) and the MMUs cannot relocate zero-page and the stack to the new blocks. I believe that, with more logic, this last should be possible, but I don't think it's worth it. When common memory is at the low end, as is usually the case, the block pointers have no effect, and zp and stk are always in RAM 0. When common memory is only at the high end, or switched out altogether, each bank has its own zp and stk. Since zero page and stack relocation is a tricky feature under normal circumstances, I've chosen to keep the hardware simple and accept the block 0 and 1 limitations.

For the more technical: The 128's designers used a 74F32 for U9 because they were concerned about the speed with which the various signals occur. (The "F" stands for Fast.) The MMU takes time to determine which (if any,) /CAS to generate. The PLA, based on the signals it receives from the MMU and elsewhere, takes time to generate its signals. Although this amounts to a few dozen nanoseconds (billionths), /RAMCASn must occur within a rather brief "window". In figure 1, Z2 replaces the actions of U9 and the delays through Z2 are not much greater than those through two 74F OR gates. Adding decoders and such before or after U9 would have increased the delays and would not have worked. (However, a similar strategy DOES work when expanding a RAM Expansion Unit, but that is another story, and will not be covered here.)

As for expanded system RAM itself, using eight 256K x1 DRAMs would have required the removal of the existing DRAMs, something I was unwilling to do. And, since the MMU selects RAM in 64K chunks, it also would have meant somehow translating the /CASs from the MMUs into extra address bits and multiplexing them into a single bit. This is not feasible, because by the time the MMUs figure out which /CAS to generate, /RAS has already occurred and by then it's too late for the new address bit to have had a valid ROW value. For the same reason, two 256K x4 DRAMs could not be used, either. The way the VIC chip refreshes memory further precludes the use of 256K x4s.

Disassembly

When your modification is completed, to test it, you will need your power supply, disk drive and monitor. Remove your printer and all accessories if they occupy too much space in the area you will be working in. Protect any finished surfaces with layers of old newspaper.

KEEP YOUR DISKS FAR AWAY FROM YOUR SOLDERING IRON!

Low-profile C128 disassembly is as follows: Begin by disconnecting all peripherals and the power supply from your computer. Remove six screws from the bottom of the case, noting the different size (a "torx" driver may be needed for some C-128 computers). Partially lift the top cover and remove the three wire LED power indicator on the left side (when you reconnect it, the polarity won't matter). At this point, you might want to discharge any static charges from yourself by touching a cold water or radiator pipe. Unscrew the keyboard grounding strap on the right side of the case. If you have enough room to work in, you may keep your keyboard connected to the main board and attached to the top cover. If not, carefully remove the keyboard connector. Do this by gently rocking it from side to side while maintaining an upward pull. This connector is "keyed" and will go back on only one way. If any of the pins bend, you can gently bend them back to the vertical position. If any pins break, you will lose use of your keyboard. When the time comes to

test your modification, I do not recommend that you repeatedly connect and disconnect the keyboard. Therefore, you might want to remove it from the top cover. Don't lose any of the little plastic "braces." If you think it will help during re-assembly, make a little sketch of how everything fits together before you start removing screws.

Remove the screws holding the RF shield in place, again noting any difference in sizes. The metal shield is soldered to the system board at one point. This will have to be unsoldered, using a solder vacuum and/or solder braid. Gently unbend the small metal tabs holding the top and bottom metal covers together and expose your computer's innards. The white gooey stuff on the larger chips and the underside of the top metal cover is heat-sink compound. Be careful not to get any on clothing, carpets, etc. You will not have to remove the circuit board from the bottom case.

To disassemble the 128D: Begin by disconnecting all peripherals and the power supply cord from the outlet and from your computer. Remove the six screws from the cabinet top by removing two screws from the bottom front and three from the rear. Slide the top back 1/2 inch and lift. Label all connectors so you can put them back from where they came. Make note of polarities! (Again, a sketch of how things WERE before disassembly may be helpful.) To remove the disk drive, disconnect connectors CN12, CN14, CN15 and CN17. Pull the lever knob from the front of the drive. Remove one screw from the left and two from the right sides of the drive. Slide back and remove. To remove the power supply, remove the screw holding the LED to the front panel. Disconnect connector CN7. Remove two screws from left, two from rear right, and one from front right. Lift. To remove the circuit board from the case, remove the screw holding the drive LED to the front panel. Remove one screw from right side and one from rear bottom. Remove eight screws from connectors CN2, 3, and 4. Unscrew the spacer the power supply rested on. Remove seven screws from circuit board and lift (Phew!).

Support Circuitry

C-128D owners should have no trouble finding room for a small perforation board to hold ICs Z2, Z3, and Z4. Owners of low-profile 128s will have to use the upside-down chip technique: the ICs are stuck to the main board upside-down and the necessary connections made by wire-wrapping directly to the pins. C-128D owners can use this method as well.

Before sticking any chip down, place a small adhesive label to the bottom and write its designator on it, and a dot to indicate the location of pin 1. Use a small piece of double-stick foam tape to hold each chip in place. This is considerably stickier that double-stick masking tape. Adhesives such as epoxy or cyanoacrylate (Krazy Glue,) SHOULD NOT be put directly in contact with the main board. Rubber and contact cements are OK, but they require "coaxing" to stick to the ICs. I positioned Z2 near U9 (near the memory chips,) Z3 and Z4 near the MMU(U7). Figure 2 illustrates my low-profile installation. This is only a suggested layout. See "Mechanical Alternatives" for other possibilities. The objective is to keep the leads as short as possible. "Ground" is available at pin 10 of U62 or the heavy trace by pin 34 of the MMU. Z2, not shown, can get +5v and Gnd from pins 14 and 7 respectively of U9. When completed, the longest wire should be the one connecting pin 12 of MMU-2 to pin 3 of Z2. This wire should not pass over the top of the MMUs.

When wire-wrapping to bare IC pins, I suggest that you strip each wire much less than you would for a normal wire-wrap. No more than 1/2 inch of bare wire should be exposed. Use a hand-wrapping tool, rather than a battery-powered gun. The wire should be slightly "backed out" of the tool. Hold the wire firmly while rotating the tool gently. The insulated portion should not become wrapped. You might want to practice on a "junk" IC before committing yourself.

Wire Z2-4 up to but NOT including the MMU connections and the cutting of resistors R29 and R30. Refer to the Signal Location table accompanying figure 1. Be sure to use the correct list for your machine("Flat" or 'D). C-128D owners should remember to use the U) pin numbers in parentheses in the schematic.

Piggy-Backing

The additional dynamic RAMs and second MMU will have to be "piggy-backed" to the old. Brand new ICs come with the pins splayed out, not at right angles to the body of the chip. This makes for a tight fit when installed by auto-insertion manufacturing equipment, but makes installation by hand difficult. For piggy-backing, to ensure good electrical connections between the pins of the upper and lower ICs, the pins of the upper chip must be straightened.

If you have a tool known as (what else?) a pin-straightener, the process is simple: stick the chip in the tool and squeeze. If not, straightening the pins one by one with pliers is NOT practical, especially for the MMU, which has 48 pins.

Begin with the DRAMs. Electrically, MOS ICs are more rugged now than they were when the technology was first invented. Still, it can't hurt to ground yourself when handling these ICs, but -- DO NOT GROUND YOURSELF AROUND "LIVE" EQUIPMENT!!! Look at each chip on-end to see how badly splayed out the pins are. Then lay the chip on its side, on a smooth, hard surface, and holding the chip by its ends, with the pins firmly against that hard surface, GENTLY "lean on it". I don't know how else to describe the process.

The goal is to bend all the pins simultaneously, by the same degree. Again, you might want to practice on a junk IC.

When you're satisfied that both rows of pins are reasonably perpendicular, slip the chip over a similarly sized IC on the main board to test the fit. Once this is acceptable, you can use pliers to bend outward the pins that will not be in contact with the lower IC. Don't try to make them horizontal. Bend them out just enough to clear the lower pins. C-128D owners need to bend out ONLY pin 16 of the DRAMs. low-profile owners, see the text below. With your soldering iron, "tin" the skinny parts of the pins that will be in contact with the lower ICs. Repeat this straightening, bending and tinning process for all the DRAMs. Don't solder them in just now. Set the prepared DRAMs aside and straighten the pins of your "extra" MMU. Before you check its fit over the socketed MMU (U7), remove any heat-sink compound from the original. Then clean your hands. If you get any of that stuff on the pins, it will be impossible to get a good, solid solder joint.

Remove the socketed MMU and decide which MMU will be the bottom one (MMU-1), and which will be the top (MMU-2). It really doesn't matter which is which.

Check the fit of the two MMUs again. Now bend outward (GENTLY!) the specified pins of what will be the upper MMU; 3 through 15, 23, and 41 through 48. Note the notch that indicates which end is pin 1. (For 40-pin and larger ICs, I put a drop of White Out or Liquid Paper beside each "5" pin -- 5, 10, 15, etc. That way, I'm not always counting pins from one end or the other. Remember, the MMU has 48 pins. Solder a wire to the wide portion of pin 23 of what will be the lower MMU. This will be the A6/7 signal to ICs Z3 and Z4. C-128D owners can skip this step and instead obtain A6/7 at pin 8 of U54, a 74LS32. In the flat 128, this signal is at pin 6 of U54, but that IC is at the left side of the video box, quite some distance away. Tin the slender portions of the pins of the upper MMU that will be in contact with the lower MMU. Place the second MMU atop the original, making sure the notches are at the same end, and slip thin cardboard, manila or pieces of a match-book cover, between the two ICs, at the ends, so that there is some space between them. Solder the skinny part of the upper pins to the wide part of the lower. This is another point where you may want to first experiment on junk ICs. Avoid excessive heating; do not linger on any one pin for an extended length of time. Once cool, all solder joints should be shiny. Dullness indicates a "cold" and possibly intermittent solder joint. Confirm that the connections are intact before inserting the MMUs in their socket. Once the MMUs are socketed, soldering to the pins risks having solder flow down into the socket, making removal of the chips difficult. Except for R/W, all needed MMU connections will be made to the "shoulders" of the bent-out pins of the upper MMU.

Now you are ready for the DRAMs.

Memory - C-128D

For the 128D, DRAM installation is straightforward. Slip the prepared DRAMs over U38-U41 on the main board, making sure the notches are in the same direction, and solder all pins except the bent-out pin 16. Solder a wire from pin 16 of the chip on top of U38 to pin 16 of the chip on top of U39. These chips are NOT next to each other, so the wire will "leap-frog" U40. Do the same thing for the DRAMs atop U40 and U41. Slip "spaghetti" on the leads of the 68 Ohm resistors and solder one end of each to the two pairs of pin 16s. (Or, snip off the excess and use heat-shrink tubing or electrical tape to insulate the wires you splice to the resistors). The free ends of the resistors go to pins 9 and 10 of Z2. Snip the U9 ends of resistors R29 and R30 on the main board (the ends toward the rear,) and connect these to pins 13 and 14 of Z2. Note that the order shown in figure 1 is correct. Z2 output Y1 selects RAM 1 and Y2 selects RAM 0; outputs Y5 and Y6 select RAM 3 and RAM 2 respectively. This is significant only if you run into trouble, and need to determine which block is malfunctioning.

Memory - Low profile

For the low profile 128, I found the prospect of piggy-backing 16 64K-by-1 DRAMs unappealing, so I used 4 64K-by-4s. In figure 3, you will see that, even though the 64K-by-4 is longer, when aligned as shown most of the signals line up properly. It doesn't matter that the Address bits are numbered differently. So, if we bend out pins 1-3 and 15-18, and let pins 1 and 18 over-hang the existing DRAMs, the rest of the signals will be taken care of by piggy-backing. (Note that +5v and ground are at positions the reverse of ordinary logic chips. Note further that the 64K-by-1 has separate Data in and out pins, which are wired together in the flat 128, and the 64K-by-4 has bi-directional Data pins -- which are numbered beginning with "D1" instead of "D0.")

Figure 4 shows my DRAM installation and part of its wiring. The easy part. Note that, because the chips are so close together, pins 2, 3 and 15-17 of the added DRAMs are clipped short. Pins 1 and 18 of the upper ICs are daisy-chained and connect to pin 16 of U49. This takes care of Gnd and /OE. The /CAS lines and 68 Ohm resistors are dealt with in the same manner as described above for the 128D. All that remain are the Data lines.

Solder wires from pins 2, 3, 15 and 17 of the chip atop U53 to the same pins of the chip atop U52. Wire each of these to each pin 2 of U42 through U45. This takes care of Data bits 4-7. Wire the same pins of the DRAMs atop U50 and U51 in the same way, with the final connections to each pin 2 of

U38-U41. And that takes care of bits D0-D3.

This "daisy-chaining" of the data lines will be much easier if the wires are prepared in advance. Cut short lengths of wire-wrap and strip the ends to leave about 5/8 inch of insulation. Tightly "pigtail" splice a longer wire to each short wire to provide the connection to pin 2 of each of the front row of mother board DRAMs. Solder the splices, then clip off the excess and you have a pre-tinned assembly to solder to the appropriate DRAM pins. The mother board DRAMs in the low-profile 128 are rather close together and numerous p.c. traces run in between the pins, providing many opportunities for the creation of short circuits. Take your time, and inspect your work often.

Not shown in figure 4 are the bypass capacitors on the main board, between the two rows of DRAMs. These make for slightly cramped quarters for the piggy-backed chips. Why not use the front row of DRAMs, you ask, where there is more room? Because the R.F. shield slopes in the front, to accommodate the keyboard, and will not clear double-high chips. Even if you dispense with the shield, the keyboard might not clear the piggy-backed chips if they were mounted along the front row. So sixteen 64K-by-1s would not have fit anyway!

Completion

Insert the double MMU into its/their socket and make the remaining connections shown in figure 1 between it/them and Z2, Z3 and Z4. Signals D6 and D7 to Z3 could be obtained at pins 41-42 of the lower MMU, but these will be difficult to get at once the two MMUs are piggy-backed. See the Signal Location table accompanying figure 1.

Testing

Check and re-check your wiring several times. Having come this far, it would be tragic to let the simplicity of the schematic mislead you into thinking that mistakes cannot be made. Make sure no little bits of wire or blobs of solder are left on the circuit board. Reassemble your machine as much as necessary to make it operable. This is where things can go wrong, so use caution. Turn your monitor on first so you won't have to wait for it to warm up. A blank screen can be very disconcerting, so make sure the monitor is in the correct mode with respect to the 40/80 key. If, when you turn on the computer, you fail to get the sign-on screen, immediately power down and find your mistake. Are the right pins of MMU-2 in contact with MMU-1? Are the right pins bent out and NOT in contact? Check the DRAM wiring. Did you snip the correct ends of R29 and R30? Did you use the correct Signal Locations for your machine? (See "Troubleshooting" at the end of this article.)

When you get a normal start-up -- complete with an attempt to boot a disk -- you'll see that the "bytes free" message still says "122365." That's built into ROM and remains unchanged by the modification. Run the short program from the Introduction. If you don't get

```
0  1  2  3
```

the modification still isn't right.

For a more convincing test, use the Machine Language Monitor to fill the same locations, say, $n8000-n80FF (where n = 0 to 3), in the four banks with different values, then examine them with "M".

Disassemble $FF05-FF44. All four banks should be identical. The last six bytes of Ram, from $FFFA to $FFFF should contain 05 FF 3D FF 17 FF in all four banks. If you use the MLM "C" command for this, ignore the "?" it prints when done. If locations $3FFF5-3FFF7 (RAM 3) contain 43 42 4D, the codes for the characters "CBM", then it isn't really RAM 3; only RAM 1 should contain that string.

Use your 128 disassembled for a few days, to confirm that all is well. If your 128 is low-profile, before you replace the R.F. shield, put masking or electrical tape on the underside, above the upside-down ICs. They should not hit, but 'tis better safe than sorry. Flatten the heat-sink tab for the MMU and put tape around the edges of the opening. If you're handy with tin-snips, you can cut off the tab and widen the opening.

You can dispense with the R.F. shield, but if any neighbors complain of interference with TV, radio, or telephone, you may find yourself in conflict with government regulatory agencies, which is never pleasant. In general, if your computer does not disrupt YOUR over-the-air (NOT cable,) TV reception, it should not bother your neighbors. If in doubt, ask them!

Software

Program 1 tests a few features of the modification. It BLOADs four Doodle format pictures to each of the four banks, demonstrating that BLOAD can access the new blocks. (Except for the load address, Doodle pictures are completely compatible with the 128's hi-res format. If you do not have Doodle, substitute the names of any 40-column hi-res 128 format pictures you may have. Multi-color pictures such as Koala CANNOT be used for this demo.) The program rapidly displays the four pictures via a simple POKE, altering the VIC/DMA block bits of $D506/54534. (If you see only two pictures, you have only RAM 0 and 1.) Halt the program by pressing any key. Adjust X in line 240 to change the speed.

Restart the program with RUN 220.

The picture in RAM 1 occupies 9K of variable space. Since this program has few variables, there is little chance of them bumping into the picture.

While it is true that the same thing could be done on a C64, by selecting different 16K video blocks, you'd be hardpressed to find the room for four pictures and a program that could do anything more than display them. (In 128 mode, the presence of the MMU registers at $FF00-04 complicates the use of the "last" 16K video block from $C000-$FFFF.) Note that while it's easy to display a bit-map in any RAM block, the Basic graphic commands only work in RAM 0.

Program 2 is the Basic loader for a relocatable Bank-to-Bank memory mover. If MOVE is where the routine is located, it can be called from Basic with

SYS MOVE, source bank, destination bank,,, source start, source end + 1, destination start

The 128 must be in BANK 15. The three commas must be present. Source and Destination Banks are 0-15. Source end + 1 must be greater than source start or you'll receive ?ILLEGAL QUANTITY. Upon return, Carry will be 0 if the move was complete. This can be determined by reading the status register into a variable with something like RREG,,,SR: IF SR AND 1 = 1 THEN the move was incomplete. The routine will not let the destination reach page $FF, and thereby clobber the MMU registers or the IRQ, NMI and other routines and pointers. However, the destination can begin in page $FF, if you're the adventurous type. In that case, if the number of bytes being moved causes the destination to "roll over" to zero page, the move will be halted and Carry set to 1. There are no restrictions on the source address, except that the last byte of memory cannot be moved since, with regard to addresses, 65535 + 1 = ILLEGAL QUANTITY. (The same restriction, by the way, applies to BSAVE.)

If you need a non-standard configuration (like RAM 2 with I/O,) POKE the source and destination CONFIGURATIONs (NOT Bank #s,) in 206 and 207 respectively, and call the Mover with SYS MOVE + 25,,,,, srce start, srce end + 1, dest start. These zero page locations are used as pointers for PRIMM which, as far as I can tell, is the only use the system makes of them. To utilize the Mover in machine language, store the srce and dest CONFIGs in $CE and $CF, store srce start in low-byte high-byte form in $AC-AD, srce end + 1 in $AE-AF. Then LDY and LDA with the low and high bytes of dest start and JSR MOVE + 47.

The Mover doesn't disable interrupts, and will not interfere with IRQ-driven activities, such as split-screens and sprite motion. It owes its small size to the use of several system routines. It owes its speed -- which isn't exactly blazing -- to the fact that two system routines are NOT entered via the Kernal jump-table. When called the prescribed way, INDFET and INDSTA each call GETCFG to convert the Bank number in .X to a configuration value. Two conversions for each byte moved proved to be too time consuming. The Mover performs the bank-to-config conversions just once, at the start of the routine, and saves them in $CE-CF. The routine later picks these up and calls INDFET and INDSTA at their common memory resident locations.

Locations $AC-AD and $AE-AF are used as start and end pointers for the source. A Kernal routine at $EEB7 is used to check that end is greater than start. Later on, to save time that would otherwise be spent in JSR and RTS, the Mover does its own comparison to determine when the end has been reached. The Basic ROM routine at $880F examines Basic text for a comma and evaluates the expression which follows to a two-byte value, 0-65535, and ReTurnS with the low-byte in .Y and the high-byte in .A.

Program 3 is an adaptation of the "Globe" animation demo on the 1764 REU disk. You need a copy of that disk for this program to work. Like the original, this program BLOADs more than thirty compressed hi-res images. The original program de-compressed the pictures and stashed them in the REU. Since there isn't enough room for that many bit-maps in the slightly less than 128K of RAM 2/3, this adaptation keeps them compressed until just before they're displayed. The previously described Mover transfers the compressed data from RAM 2/3 to a graphic area in either RAM 0 or RAM 1. Then they're de-compressed and bits 6/7 of the Ram Configuration Register at $D506/54534 are changed to display the picture. I call this sort of animation, moving a full bit-map, "brute force." For "Globe" (and especially "Pound", which is multi-color,) it works only because the colors are static and unmoving.

Although the machine language program "compress.bin" was intended for 64 mode, when located in the same RAM block as a bit-map, it works fine in 128 mode. (It is BLOADed twice to put it in both RAM 0 and 1.) It assumes the bit-map is at $2000/8192. SYS 49155 uncompresses and SYS 49152 compresses a bit-map. PRINT PEEK(253) + 256 * PEEK(254) then gives the end address of the compressed bit-map data.
Colors are not affected.

The demo sets aside 36 2304-byte buffers -- 18 each in RAM 2 and 3 -- to hold the compressed data. There is room in each bank for 27 such buffers. That size represents the largest .cmp file in disk blocks. For some reason, when Globe files are de-compressed, some debris is left in the first 768 bytes of the bit-map. Line 400 clears that by using a short-coming of the Mover as a feature: When source and destination over-lap in the same Bank, the move becomes a fill. Delete line 400 for "Pound." (Which explains the need for the seemingly redundant BANK 15 in line 420.)

Suggestions (software)

Program 1 and the Mover can be combined for a "brute force" form of hi-res animation. Banks 2 and 3 can each hold up to 6 hi-res pictures. You could move a picture into the graphic area of RAM 0 and while it is being displayed, move the next picture into the same area of RAM 1, then change the VIC/DMA bits to display that RAM. Move the next picture into RAM 0, etc. Multi-color pictures have the additional complication of Color Memory.

The capacity of Banks 2 and 3 equals 2/3 of a single-sided floppy. While this is not as spacious and cannot be as fast as a 1750 REU, it's still nothing to turn up one's nose at. Someone out there should be able to coax Geos 128 into utilizing the new Banks, alleviating some of its tedious disk activity.

On the Basic side, the Mover could form the basis of a RAM-drive. Or it could completely stash Banks 0 and 1 in 2 and 3, saving your current program and all its variables while you load and run a second program, then return to where you left off. The Mover would need some modification to do this though.

To determine compatibility, the Intro program, or its MACHINE LANGUAGE equivalent, could determine if a 128 really has Banks 2 and 3 or is non-modified.

And what could CP/M do with 256K of memory?

Suggestions (hardware)

I do not anticipate any software incompatibility with this modification. But should any arise, a two position switch can be installed to cause the machine to appear to be non-modified. When the switch connects input "C" of Z2 to MMU-2, Banks 2 and 3 are available. When the switch is in the Gnd position, the new Banks disappear, and accesses of them default to RAM 0 and 1, as in a non-modified C128. Use a double-pole switch. The other pole might come in handy.

JiffyDOS and others...

For compatibility with JiffyDOS -- and perhaps other disk speed-ups -- combined with (some) serial printers, it may be necessary to make a small addition to the system mother board. Connect a 1000 Ohm 1/4 Watt resistor between +5 Volts and "FSDIR" -- the line from MMU-1 that tells the Fast Serial circuitry which way the data is going. This will prevent spurious FILE NOT FOUND errors when JiffyDOS 128 and the memory expansion modification are both enabled. This problem occurs ONLY in 128 mode when the printer is on AND the 128 is in SLOW mode ??? FSDIR is available at pin 4 of U58 (74LS03) in both the low-profile and the 'D. Note that the IC designators on some 128s are not very readable. The slashed zero can easily be mistaken for an "8."

How does the pull-up resistor eliminate FILE NOT FOUND? I wish I knew. JiffyDOS does not use the Fast Serial capability, but I can see (vaguely,) how a serial printer might be involved in the problem. C64 mode automatically disables the Fast Serial circuitry, so any glitches therein would have nowhere to go. But why does FAST mode, specifically, the blanking of the 40 column screen, eliminate the problem?

One problem may remain: In 128 mode, the JiffyDos wedge command (Load&Run) sometimes doesn't. This might be caused by a bug in the wedge. With JiffyDOS enabled but its wedge disabled, and a 128 Dos wedge from Compute!'s Gazette enabled, the (Load&Run) command works fine. Run"prog.name" ALWAYS works. So does JiffyDos's (Load&Run) in C64 mode ?

Troubleshooting

As is usually the case in a project such as this, the most common causes of it failing to work are wiring errors and bad connections; intermittent or "cold" solder joints and short circuits caused by solder bridges. The chances of shorts causing any permanent damage to the 128 are minimal. The difficulty is in locating them.

If the 128 powers up "dead" -- blank screen, no drive activity -- the problem could be anywhere, but you may not have to look everywhere. Look for wiring errors in the enabling of the DRAMs serving as RAM 0 and 1 (Z2) or shorts on the address or data lines of those DRAMs as a result of the piggy-backing. If the 128 powers up with a garbled screen, characters that don't look right, or no disk activity, the two MMUs are probably interfering with each other. Check the MMU piggy-back connections and the wiring of Z3/4. If you run the multiple DOODLE program and the pictures in RAM 0 and 1 are fine, but those in the added RAM(s) are complete garbage, check the wiring of Z2 and the two MMUs. If the pictures in additional RAM are partially garbled, one or more address and/or data connections to those RAMs might be missing. This could be caused by poor piggy-back connections. (Remember, in the low-profile it doesn't matter if the address/data bits to added RAM are "scrambled" -- not in order -- as long as you have all of them and the scrambling is consistent.) Examine those banks with the MLM .m command. If bytes change every time you display them, that's a sure sign that one or more address or data bits are missing. Again, shorts on the address or data lines would cause the 128 to not work at all.

Supplement

Low-profile Mechanical Alternatives or
If I knew then what I know now...

When I expanded my low-profile 128 to 256K, the second 128K, consisting of four 64K-by-4 dynamic RAMs, was added using piggy-backing. To increase the modification to 512K, eight more DRAMs were installed using more piggy-backing for the first four and a small board for the remaining four. Now that I've modified more than one "flat" 128, I've devised other strategies.

1. There is room just in front of the video box for a small perforated "daughter" board. This could hold 4 DRAMs for the 256K modification, and could be large enough to include the support chips -- except for the second MMU which should still be piggy-backed because of the many connections. The daughter board could even be large enough to include the chips needed for the 512K expansion, which could be added at a later time. The four DRAMs mounted on the daughter board would be wired to the appropriate points on the mother board. When the time came to expand to 512K, the eight additional DRAMs could be piggy-backed to the daughter board DRAMs or to the second row of mother board DRAMs. (Or any combination thereof.)

2. The method I've actually been using has been to use a small board only for the DRAMs; 4 for the 256K modification or, using piggy-backing, a total of 12 for the 512K. (The piggy-backing is as described for the 128D in the 256K plans.) The support chips are still mounted upside-down using double-stick foam tape and the necessary connections made by wire-wrapping to the pins.

Method 1 has the advantage of being more easily undone, should the need ever arise. I've found method 2 to be the fastest and least troublesome, since the only "hidden" wiring is on the underside of the DRAM board. For EITHER method, if dip sockets are used, which I STRONGLY recommend, the R.F. shield won't fit -- with or without piggy-backing! (There is enough headroom -- though just barely -- for three levels of DRAMs.) You could dispense with the shield, or make a cut-out, using tin-snips, sufficient to clear the add-on board components. The area of the mother board beneath the add-on board MUST be insulated with a layer of vinyl tape. The add-on or daughter board is held in place with small pads of double-stick foam tape.

When positioned in the logo (C=) area of the low-profile mother board, the daughter board is centrally located to all the signals it needs. As illustrated, the daughter board is tight, but workable. the dimensions given are the largest possible for the area involved. The positions of the ICs (particularly Z2,) were chosen to keep the signal leads as short as possible. Z1, the second MMU, should still be piggy-backed due to the many connections involved.

Flat ribbon-cable is handy for groups of conductors, but its stiffness sometimes makes it difficult to position. I have found "tone-arm" cable, the thin wires used inside a phonograph tone arm, very useful. (The smallest gauge is a bit too fine to work with. Choose a size similar to or slightly larger than wire-wrap, which is usually #28 or #30.) It comes jacketed and unjacketed, shielded and unshielded. The shield and jacket are easily removed and the four conductors are flexible enough to make turns that ribbon cable won't. (For a little more work and a little less flexibility, you could also assemble your own bundles of twisted wire-wrap, in different colors so you can tell them apart.) Bundles of four conductors can be assigned to lower and upper address bits and lower and upper data bits. Once everything is tested and functioning properly, a dab or two of hot-melt glue will hold the cabling in place.

Materials List

8722 MMU (*CO389 $16.95)
Available from Grapevine Group

ICs Z3 and Z4 ($ 8.00 per pair)
Available from Richard Curcio
or Parsec, Inc.

The below parts are available from B.G. Micro, the Grapevine Group and supplies are available from Radio Shack.

1) 74LS138 (B.G. Micro)
4) 64Kx4 dynamic RAMs
(4464, 41464, etc.)
150ns or faster (* call)
2) 68 Ohm, 1/4W resistors
1) 2K Ohm, 1/4W resistor
1) 1K Ohm, 1/4W resistor (for JiffyDos)
1) DPDT switch (disabling)
solder
30 gauge wire-wrap wire
electrical tape,
"spaghetti", etc.

* Grapevine Group part numbers

Richard Curcio can upgrade your computer for you. Contact Richard at the address listed for the latest pricing and shipping information. Be sure to use a self addressed stamped envelope!

Addresses of companies and persons mentioned in this article:

Grapevine Group, Inc.
3 Chestnut St.
Suffern, NY 10901
(914) 357-2424

Richard Curcio
22 Seventh Ave.
Brooklyn, NY 11217

ICs Z3 and Z4 may be obtained from Richard for $8.00 the pair for the US and Canada or $10 (US) outside North America.

B.G. Micro
P.O.Box 280298
Dallas, TX 75228
1-214-271-5546

Parsec, Inc.
PO Box 111
Salem, MA 01970-0111

Foreign orders (excluding Canada) of Z3 and Z4 MUST include, on a separate sheet, the following:

Importer's Statement:

We will import these COCOM controlled goods and will not ship them outside COCOM participating countries without prior authorization from the appropriate national authorities.

_____

(signed)

Program 1

```
100 GRAPHIC1:TRAP310:GOSUB340:BANK15
110 REM BLOAD 4 DOODLE FORMAT PICTURES INTO THE 4 BANKS
120 POKE54534,PEEK(54534)AND63:REM VIC DISPLAYS RAM 0
130 BLOAD"DDMIDDLE EARTH",B0,P7168
140 POKE54534,PEEK(54534)AND63ORCR(1):REM DISP. RAM 1
150 BLOAD"DDGRAPHIC BLOCKS",B1,P7168
160 POKE54534,PEEK(54534)AND63ORCR(2):REM DISP. RAM 2
170 BLOAD"DDLETTERS",B2,P7168
180 POKE54534,PEEK(54534)AND63ORCR(3):REM DISP. RAM 3
190 BLOAD"DDPLAN",B3,P7168
200 REM RE-READ ARRAY SO PROG MAY BE RESTARTED
210 REM WITHOUT RELOADING PICTURES
220 TRAP310:GOSUB340
230 BANK15:GRAPHIC1:C=0
240 X=199:REM DISPLAY SPEED
250 C=C+1:IFC>3THENC=0
260 GETA$:IFA$<>""THEN310:REM EXIT IF KEY PRESSED
270 POKE54534,PEEK(54534)AND63ORCR(C)
280 FORT=0TOX:NEXT
290 GOTO250
300 REM SET VIC DISPLAY FOR NORMAL
310 POKE54534,PEEK(54534)AND63:GRAPHIC0:END
320 :
330 REM READ VALUES FOR RAM CONFIG REGISTER
340 RESTORE:FORI=0TO3:READ D:CR(I)=D:NEXT:RETURN
350 DATA 0,64,128,192
```

Program 2

```
100 REM RELOCATABLE BANK-TO-BANK MEMORY MOVER
110 REM CHANGE "SA"
120 :
130 BANK15:SA=4864
140 FORI=0TO107:READD:POKESA+I,D:NEXT
150 DATA 201, 16,144,  3, 76, 40,125,224
160 DATA  16,176,249,133,206, 32,107,255
170 DATA 133,207,166,206, 32,107,255,133
180 DATA 206, 32, 15,136,132,172,133,173
190 DATA  32, 15,136,132,174,133,175, 32
200 DATA 183,238,176,216, 32, 15,136,132
210 DATA 195,133,196,169,172,141,170,  2
220 DATA 169,195,141,185,  2,160,  0,166
230 DATA 206, 32,162,  2,166,207, 32,175
240 DATA   2,230,172,208,  2,230,173, 56
250 DATA 165,172,229,174,165,173,229,175
260 DATA 240, 16,230,195,208,225,230,196
270 DATA 240,  6,165,196,201,255,144,215
280 DATA  56, 96, 24, 96
```

Program 3

```
100 IFRGR(0)=5THEN
    PRINTCHR$(15)"*** 40 COLUMNS ONLY ***"
110 TRAP440
120 COLOR0,1:COLOR4,1:GRAPHIC0:SCNCLR:COLOR5,15
130 POKE57,0:POKE58,28:CLR:
    REM LIMIT VARIABLES TO BELOW BIT-MAP IN RAM 1
140 SLOW:INPUT"LOAD PICTURES (Y/N)";A$
150 IFA$="Y"THEN190
160 IFA$<>"N"THEN140
170 BANK0:IFPEEK(4864)<>201OR
    PEEK(49155)<>76THENPRINT"NO ML!":END
180 FAST:GOSUB480:GOTO300
190 BLOAD"COMPRESS.BIN",B0
200 BLOAD"COMPRESS.BIN",B1
210 FAST:GOSUB480:GOSUB680:SLOW
220 PRINT""
230 P=0:B=2
240 P=P+1:IFP>36THEN300
250 IFP>18THENB=3
260 PRINT"LOADING IMAGE"P"OF 36"
270 :
280 BLOAD"GLOBE"+STR$((P-1)*10)+".CMP",B(B),P(SS(P-1))
290 GOTO240
300 COLOR1,10:GRAPHIC1,1
310 REM COPY CLR'D & COLORED GRAPHIC AREA TO RAM 1
320 BANK15:SYSMOVE,0,1,,,7168,16384,7168:SLOW
330 :
340 P=36
350 P=P-1:IFP<0THENP=35
360 REM MOVE COMPRESSED PIX TO BIT-MAP AREA
370 BANK15:SYSMOVE,SB(P),DB(P),,,SS(P),SS(P)+SZ,BM
380 BANK(DB(P)):SYS49155:REM DE-COMPRESS IT
390 REM CLEAN IT UP (USE MOVER TO FILL 1ST 3 PGS
    W/ZEROES) * GLOBE ONLY *
400 POKEBM,0:BANK15:SYSMOVE,DB(P),DB(P),,,BM,BM+767,BM+1
410 REM CHANGE MMU RCR BITS 6-7
420 BANK15:POKE54534,PEEK(54534)AND63ORVB(P)
430 GETA$:IFA$=""THEN350
440 POKE54534,PEEK(54534)AND63:GRAPHIC0
450 END
460 :
470 REM INITIALIZE VARIABLES & ARRAYS
480 SZ=2304:A=1024:REM SIZE, START
490 MOVE=4864:BM=8192:REM MOVER, BIT-MAP
500 DIM VB(35):REM VIC  BANK VALUES
510 DIM SB(35):REM SRCE BANK FOR MOVER
520 DIM DB(35):REM DEST BANK FOR MOVER
530 DIM SS(35):REM SRCE START ADDRESSES
540 :
550 REM FILL VB() & DB()
560 FORI=0TO34STEP2:VB(I)=0:DB(I)=0:NEXT
```

(continued on next page)

Program 3(continued)

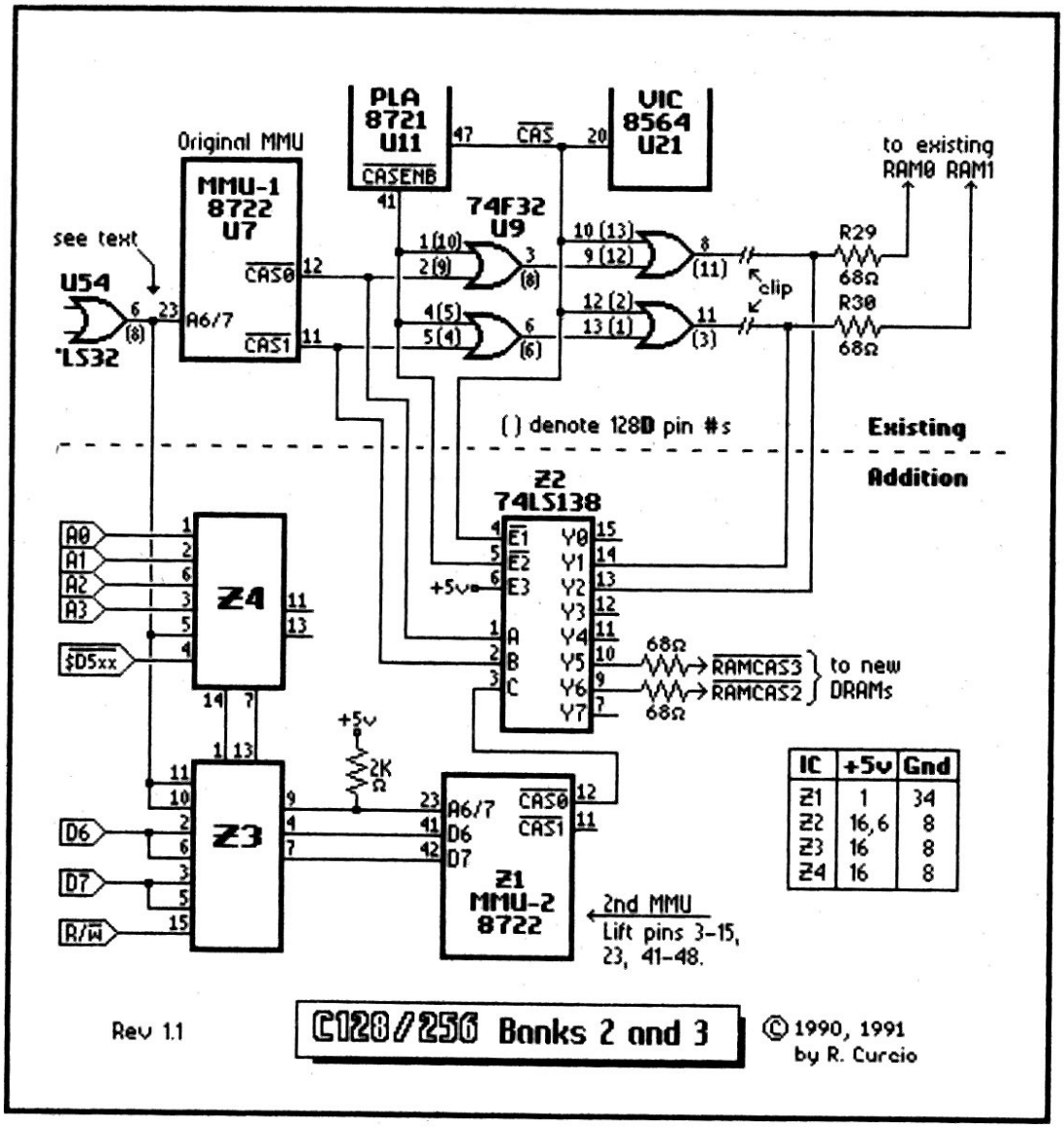
```
570 FORI=1TO35STEP2:VB(I)=64:DB(I)=1:NEXT
580 :
590 REM FILL SB()
600 FORI=0TO17:SB(I)=2:NEXT
610 FORI=18TO35:SB(I)=3:NEXT
620 :
630 REM FILL SS()
640 FORI=0TO17:SS(I)=A:SS(I+18)=A:A=A+SZ:NEXT
650 RETURN
660 :
670 REM *** INSTALL MOVER ***
680 RESTORE:BANK15
690 FORI=0TO107:READD:POKEMOVE+I,D:NEXT
700 RETURN
710 DATA 201, 16,144,  3, 76, 40,125,224
720 DATA  16,176,249,133,206, 32,107,255
730 DATA 133,207,166,206, 32,107,255,133
740 DATA 206, 32, 15,136,132,172,133,173
750 DATA  32, 15,136,132,174,133,175, 32
760 DATA 183,238,176,216, 32, 15,136,132
770 DATA 195,133,196,169,172,141,170,  2
780 DATA 169,195,141,185,  2,160,  0,166
790 DATA 206, 32,162,  2,166,207, 32,175
800 DATA   2,230,172,208,  2,230,173, 56
810 DATA 165,172,229,174,165,173,229,175
820 DATA 240, 16,230,195,208,225,230,196
830 DATA 240,  6,165,196,201,255,144,215
840 DATA  56, 96, 24, 96
```

MOVE.SRCE

```
1000 SYS4000
1010 ;
1020 ;POWER ASSEMBLER
1030 ;
1040 *= $1300
1050 ;
1060 .MEM
1070 ;
1080 ;***  BANK-TO-BANK MEMORY MOVER  ***
1090 ;
1100 SETUP  CMP #$10
1110 :     BCC XCP   ; SRCE BANK < 16
1120 ERROR  JMP $7D28 ; ILLEGAL QUANTITY
1130 XCP    CPX #$10
1140 :     BCS ERROR ; DEST BANK > 15
1150 :     STA $CE
1160 :     JSR $FF6B ; GET DEST CNFG
1170 :     STA $CF
1180 :     LDX $CE
1190 :     JSR $FF6B ; SRCE CNFG
1200 :     STA $CE
1210 :     JSR $880F ; SRCE START
1220 :     STY $AC
1230 :     STA $AD
1240 :     JSR $880F ; SRCE END+1
1250 :     STY $AE
1260 :     STA $AF
1270 :     JSR $EEB7 ; COMPARE START & END
1280 :     BCS ERROR ; START>END
1290 :     JSR $880F ; DEST START
1300 :     STY $C3
1310 :     STA $C4
1320 :     LDA #$AC
1330 :     STA $02AA ; INDFET POINTER
1340 :     LDA #$C3
1350 :     STA $02B9 ; INDSTA POINTER
1360 ;
1370 :     LDY #$00
1380 MOVIT  LDX $CE   ; GET SRCE CNFG
1390 :     JSR $02A2 ; DO INDFET
1400 :     LDX $CF   ; GET DEST CNFG
1410 :     JSR $02AF ; DO INDSTA
1420 :     INC $AC   ; INCRMNT PNTR
1430 :     BNE CSE
1440 :     INC $AD
1450 CSE    SEC
1460 :     LDA $AC
1470 :     SBC $AE   ; COMPARE TO END
1480 :     LDA $AD
1490 :     SBC $AF
1500 :     BEQ EXIT  ; DONE
1510 :     INC $C3
1520 :     BNE MOVIT
1530 :     INC $C4
1540 :     BEQ HALT
1550 :     LDA $C4
1560 :     CMP #$FF
1570 :     BCC MOVIT
1580 HALT   SEC       ; ROLLED OVER
1590 :     RTS
1600 EXIT   CLC       ; COMPLETE MOVE
1610 :     RTS
1620 ;
```

CHANGES FOR "POUND.DEMO"

```
120 COLOR0,16:COLOR4,16:GRAPHIC0:SCNCLR:COLOR5,7
240 P=P+1:IFP>32THEN300
260 PRINT"      LOADING IMAGE"P"OF32"
280 BLOAD"POUND"+STR$(P-1)+".CMP",B(B),P(SS(P-1))
300 COLOR1,11:COLOR2,3:COLOR3,7:GRAPHIC3,1
340 P=0
350 P=P+1:IFP>31THENP=0
```



Signal	Location "Flat"	Location "D"
$D5xx	U3, pin 14	-same-
A0	U10, 30	"
A1	U10, 31	"
A2	U10, 32	"
A3	U10, 33	"
R/W	MMU, 32	"
D6	U13, 16	U13, 5
D7	U13, 18	U13, 9
CAS0	U9, 2	U9, 9
CAS1	U9, 5	U9, 4
CASENB	U9, 1	U9, 5
CAS	U9, 10	U9, 2
A6/7	see text	U54, 8

Notes:

Resistors are 1/4 Watt. Values non-critical.
Use 39-82Ω for 68Ω, 1K-4.7K for 2KΩ.
Leave unused IC pins un-connected.

Figure 1

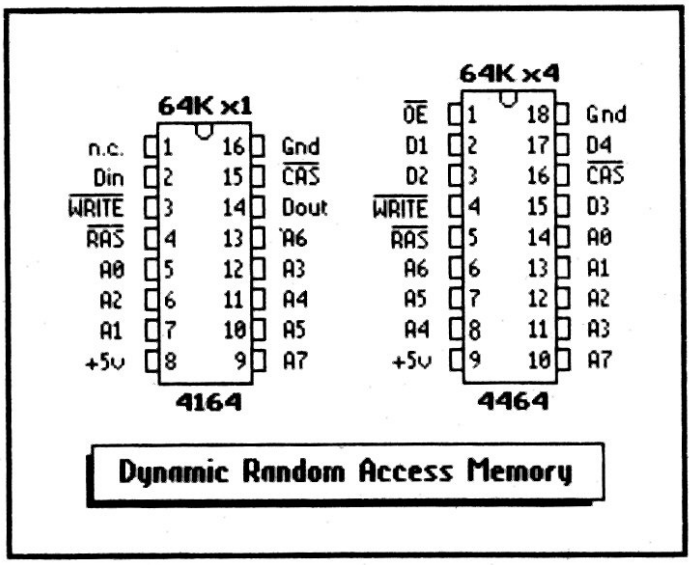


Figure 3

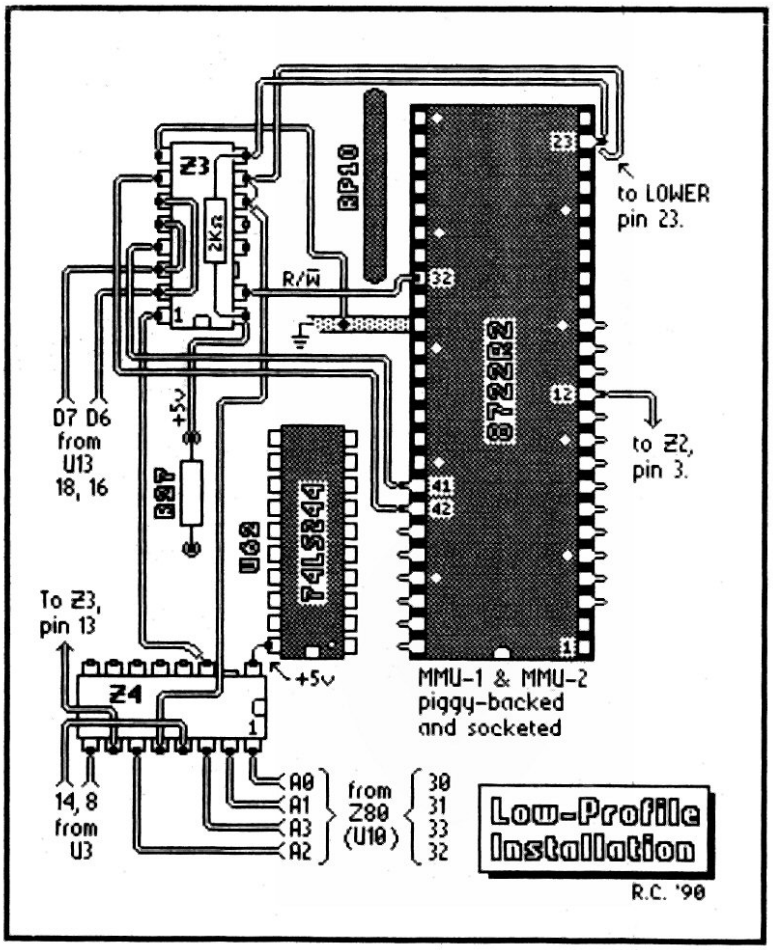
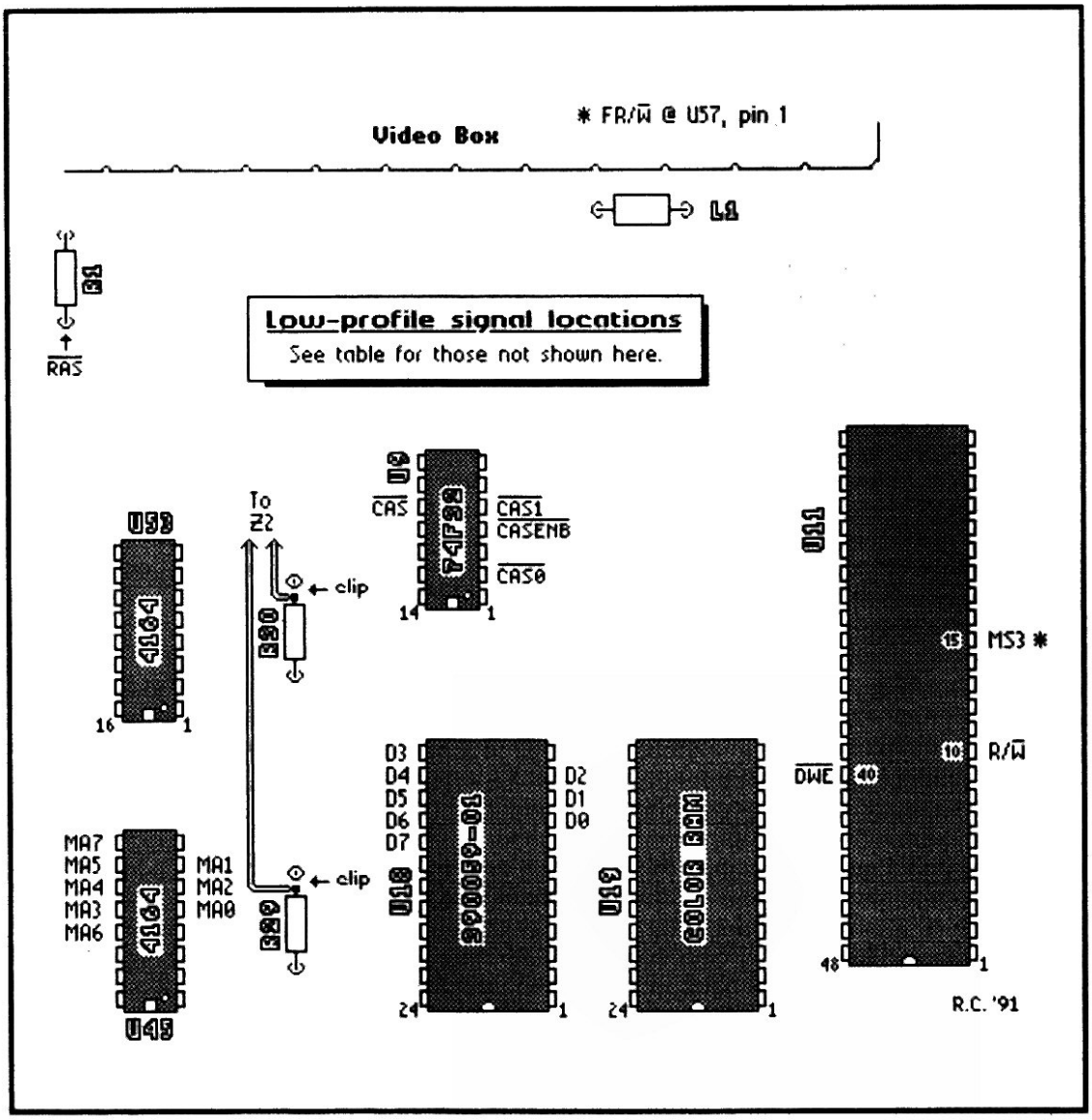


Figure 2

Notes:

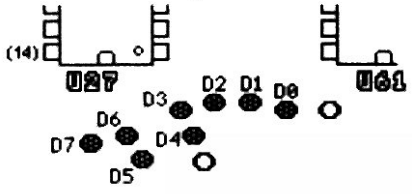
Z3 and Z4 upside-down. Z2 not shown.
U13 pin #s for flat 128 ONLY.

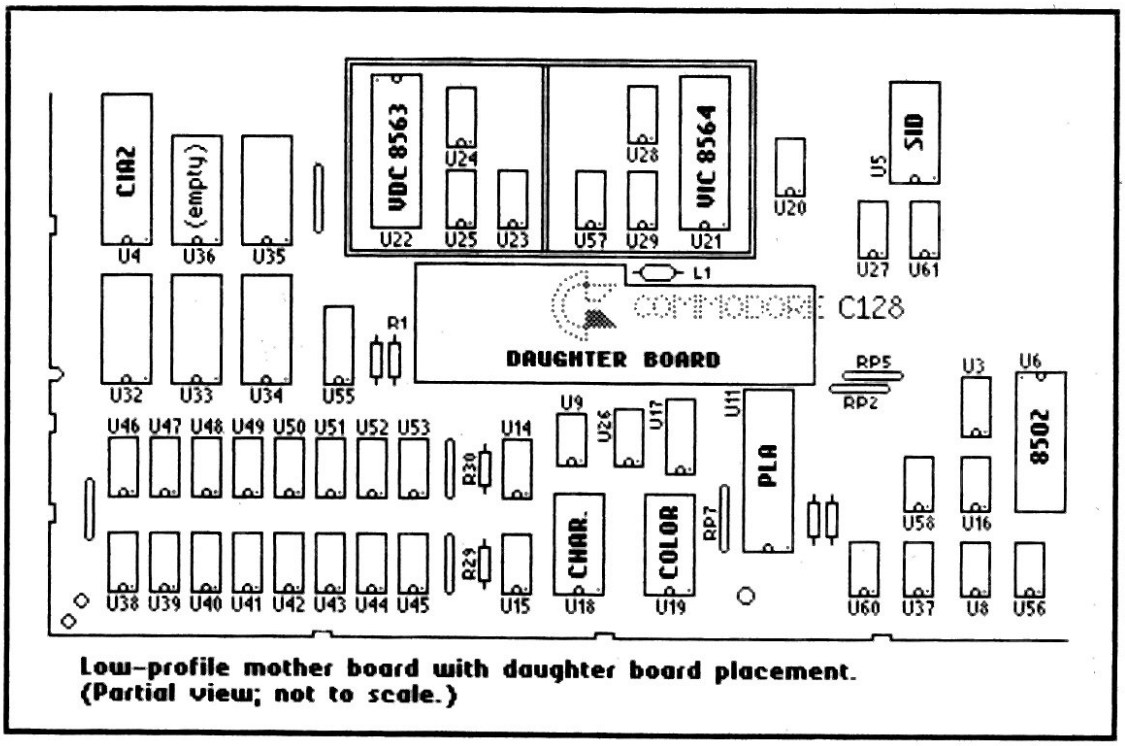


MA0-7, D0-7, /DWE and /RAS needed for DRAMs on daughter board. Be sure to take D0-7 from Character ROM U18, not from Color RAM U19.

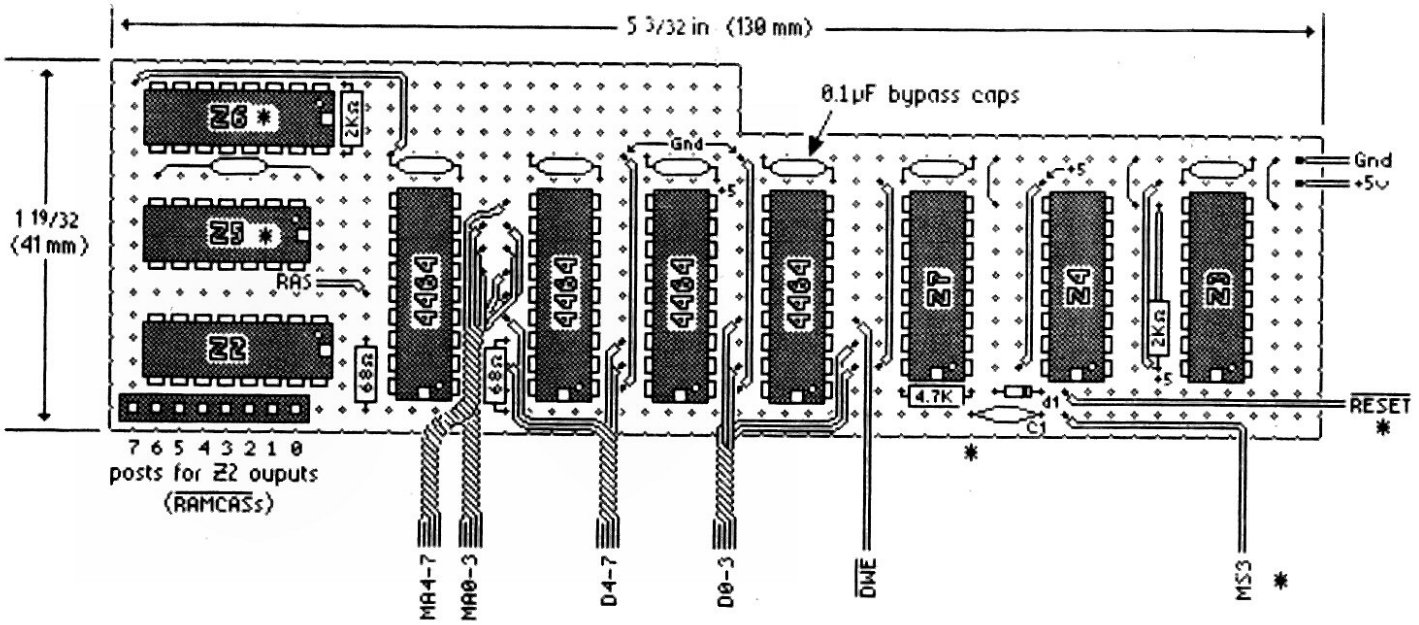
*** MS3 and FR/W needed for 512K only.**

Another place to get D0-7 on the low-profile.





Low-profile mother board with daughter board placement.
(Partial view; not to scale.)



*** ICs Z5, Z6, Z7, associated components and signals required for 512K only.**
Note reversed locations of +5v and ground for DRAMs.

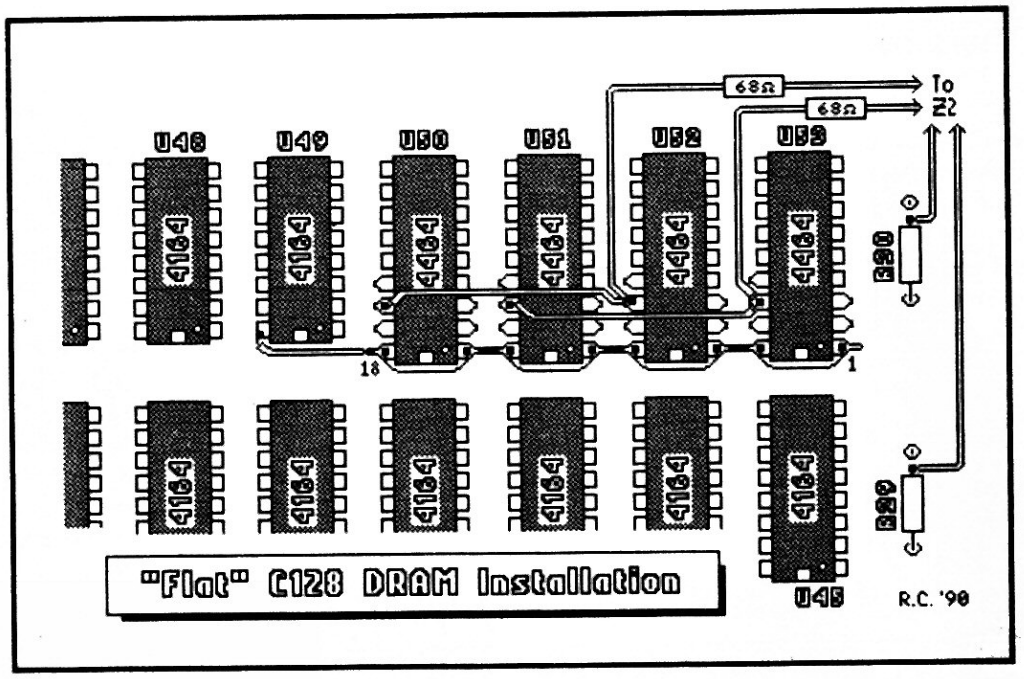


Figure 4

Piggy-backing 18-pin 64K x4s to 16-pin 64K x1s.
(Data lines not shown. See text.)

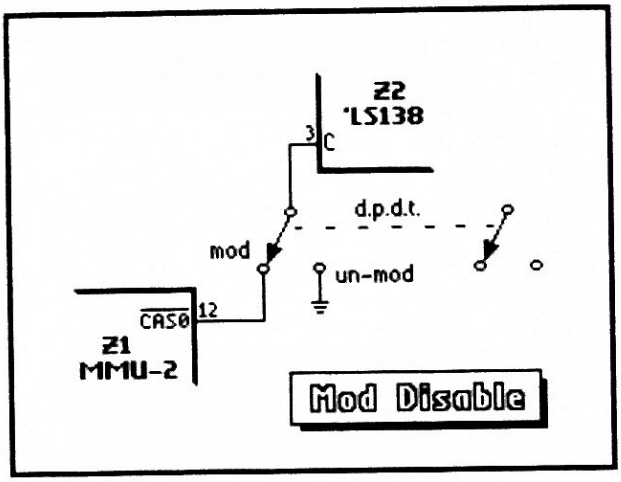
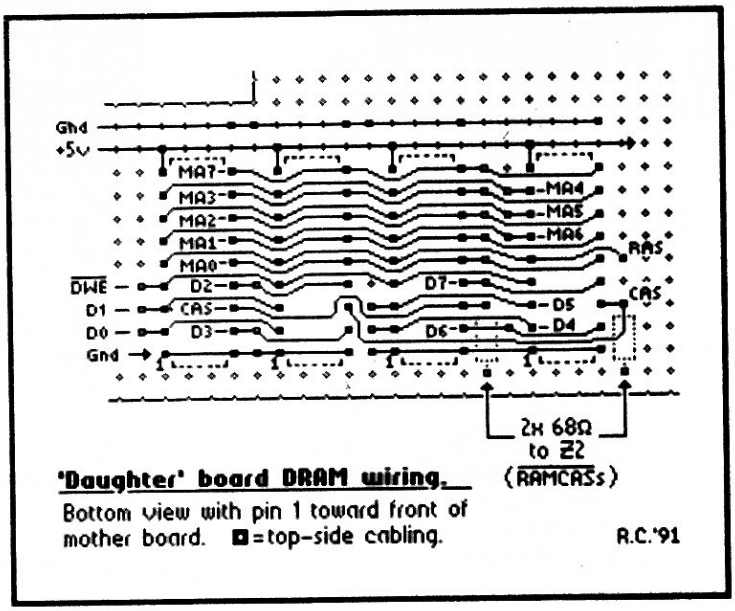


Figure 5



EXPANDING THE COMMODORE 128 *by Richard Curcio* "Part 2: 4 Mode 512K"

INTRODUCTION

Several bits in the many registers of the C128's Memory Management Unit (MMU) are unused. The MMU documentation describes some of these bits as "reserved for future expansion". Such is the case for bits 4 and 5 of the Ram Configuration Register (RCR) at $D506 in I/O space. The prevailing assumption has been that these two bits would select four "SuperBanks" of 256K each. In other words, 1 Megabyte of system RAM.

The more I pondered that possibility, the more convinced I became that providing that much memory might be impractical from both a software and hardware standpoint. But the success of my implementation of the formerly non-existent RAM 2 and 3 lead me to think that some form of SuperBanking might be possible.

This modification is an extension or upgrade of Part 1. By adding another 256K in the form of eight 64K x4 DRAMs, and 3 other ICs, a 256K-modified C128 is brought to a total of 512K with four modes of operation.

THESE PLANS ARE OF NO USE WITHOUT THE 256K CIRCUITRY. PLEASE SEE TWIN CITIES 128 ISSUE #30 FOR THE PLANS, DETAILS, PARTS LIST, ETC.

DISCLAIMER

This modification will render any warranties on your equipment null and void. The Author and Publisher do not assume any liability for Purchaser's implementation of these instructions. All information is believed to be accurate.

DIFFICULTIES

Attention "flat" C-128 owners, because of the low headroom inside a flat C-128 case this project is not for the fainthearted or inexperienced. Though it can be done, it is a VERY tight fit and requires a great amount of skill to make it fit!

The MMU has the ability to keep portions of RAM 0 "common" in all configurations. This so-called "common memory" is used to hold routines that do the actual bank switching and therefore must be accessible in all configurations. Without common memory, whenever the microprocessor attempted to change configurations, code would be switched out from under it. There are ways around this but common memory is still needed to put them into memory in the first place.

To switch "SuperBanks", then, would require "Super Common Memory" --- a portion of memory accessible in all SuperBanks. Just which of the remaining unused MMU bits Commodore intended to specify SCM is unknown. They can't be at $D506; bits 4 and 5 are the presumed SuperBank bits and the 256K modification uses the formerly unused bit 7 of the RCR to select RAM 2 or 3 for VIC/DMA access, as was intended. This implies that SuperBanking with SCM capability would have to be a two-register operation. Clearly, too much hardware and tricky software would be needed.

Instead of using the SuperBank bits to select four sets of 256K, this modification uses them to give 512K four modes.

OPERATION

The little bits/blocks table within the schematic summarizes the four modes. When bits 5/4 of $D506 are %00, the normal condition after power-on or reset, the C128 behaves as if it was "only" 256K modified. When bits 5/4 of $D506 are %01 (mode "B") RAM 4/5 can be accessed as if they were RAM 2/3 after the appropriate BANK statement. When bits 5/4 are %10 (mode "C") RAM 6/7 become the new RAM 2/3. In modes B and C, RAM 4-7 might be considered "extended memory". Note that "true" RAM 0 is always in place in these modes, so common memory remains so. When bits 5/4 = %11, RAM 4-7 becomes a second, "alternate" set of RAM 0-3, complete with its own common memory, zero page and stack. Machine language is needed to use this mode. When bit 7 of $D506 is %1, letting the VIC display RAM 2 (bit 6 = %0) or RAM 3 (bit 6 = %1) then bits 4/5 determine just which RAM 2/3 (true, extended or alternate) is displayed (see program 4, "Super 8"). This also applies to REU accesses of RAM 2/3 and to the use of the Memory Mover supplied with the 256K plans; a POKE to $D506 (decimal 54534 in BANK 15), will allow you to move data between RAM 0/1 and any of the three RAMs 2/3, but you can't move data between the three sets of RAM 2/3 -- at least, not in one operation. A "SuperMover" is needed.

Starting in normal mode, after BANK 15: POKE 54534, 20 accesses of BANK 2/3 will really access RAM 4/5. POKE 54534, 36 switches in RAM 6/7 as 2/3. To return to normal mode (1K of common memory at the low end, VIC displays RAM 0, "true" RAM 2/3), POKE 54534, 4. In some instances, PEEK, AND and OR should be used in the POKE statements so that changing bits 4/5 does not unintentionally affect the other bits of the RCR. Note that before RAM 4-7 can be safely accessed, page $FF of those blocks must be initialized. More on this under "SOFTWARE".

CIRCUIT THEORY

Integrated Circuits Z1, Z3 and Z4 of the 256K circuitry are unchanged. However, /CAS1 from MMU-1 (the original) no longer goes directly to the B input of Z2. The two /CAS0s from the two MMUs form a (fairly) straightforward 2-bit "CAS code". With a third bit for

Z2's C input, all eight Z2 outputs are utilized and we now have to detect the "no bank" condition when the MMU registers at $FF00-04 are addressed (neither of the original /CASs go low). NAND gate Z5-d takes care of that via Z2's active-high E3 Enable, which was connected to +5 volts in the 256K mod.

NAND gates Z5-a, -b, and -c combine FR/W with a signal from Z4 to provide a Clock Pulse for latch Z7. This captures the value written to bits 4/5 of the RCR. (A four-bit latch was used to allow for possible "future enhancements". Circuitry for reading Z7 is unnecessary since bits 4/5 of the RCR retain whatever is written to them and we can simply read the MMU). Diode d1 pulls Z7's Master Reset low during power-up and RESET. This clears the latch contents to zeros, establishing the default condition. Whenever the 128 goes into C64 mode, MS3 goes low. Capacitor C1 will then generate a brief low-going pulse which will also clear the latch, so we start in a known state in C64 mode.

Latch outputs Q0 and Q1 go to select bits S0 and S1 of Z6, a 74F153 dual 4-to-1 mux. (An 'LS153 apparently works -- even at 2 MHZ -- but I suspect that, with respect to timing, that might be cutting things close). The "mode" selected, routes the appropriate signals on Z6's Inputs to Z2's B and C inputs. The disable switch SW is optional, but recommended. (See footnote under "C64 MODE"). When the switch is open, the Ea and Eb enables of Z6 are held high and that chip is disabled; its Ya and Yb outputs are held low, and Z2 can therefore select only the original RAM 0 or 1 -- or neither, if Z5-d's output is low. If the switch is omitted, Ea and Eb MUST be connected to ground.

Using FR/W (called R/WA in the SAMs schematic,) to generate a WRite makes latch Z7 immune to alteration by an external DMA, such as from an REU. This duplicates the MMU's immunity to outside alterations. (FR/W also occurs a little sooner than R/W and has faster edges). The astute will recognize Z5-a, -b, -c as equivalent to an OR gate, and some might be tempted to use one of the now-unused gates from U9. I advise against that. So far, only two mother board connections have had to have been cut -- R29 and R30 -- and I'd like to keep it that way. Besides, a NAND is still needed to detect the no bank condition mentioned earlier.

MECHANICAL CONSIDERATIONS

As in the 256K modification, there should be ample room for the new circuitry inside a 128D. A small circuit board can be used to hold ICs Z5-7, or the double-stick foam upside-down wire-wrapping (DSFUDWW) method can be used. Piggy-back two more levels of four 64Kx4 DRAMs to existing memory, wiring the /CAS pins of each level the same way you did for the first level of piggy-backed DRAMs. For the low-profile 128, things are a bit more involved. Low-profile owners will almost certainly have to dispense with the R.F. shield, or cut an opening in it.

In my low-profile 128, I piggy-backed four more 64Kx4 DRAMs to U46-U49, alongside the first four atop U50-U53. The four remaining DRAMs were mounted on a small board positioned in the logo area of the mother board (which I insulated with a layer of vinyl tape), and held in place with small pads of double-stick foam. ICs Z1-Z4 of the 256K circuitry remained where they had been and Z5-Z7 were installed using the same DSFUD-WW technique. The illustration roughly indicates the layout. Latch Z7, not shown, was placed near U3 and ICs Z3 and Z4. (also see the "daughter" board suggested in the Supplement with the 256K plans). The disable SWitch was mounted on the left side of the bottom case, in the vicinity of the three-pin LED connector, CN13. I used a double-pole double-throw slide-switch (even though only one pole is needed), with a slider that didn't protrude very much, to make it difficult to actuate accidentally. The case plastic is soft, so cutting a rectangular opening was easy. A mini toggle switch has the advantage of using a round hole.

The way I installed the DRAMs probably was not the best course of action. For the low-profile, I recommend that the first four DRAMs be piggy-backed to the mother board DRAMs, as described in the 256K plans, and the remaining eight installed on a small board in the logo area. If starting from scratch, another strategy is to put all 12 DRAMs on the memory board. Without the metal RF shield, there is just enough headroom for three levels of 4 DRAMs. The piggy-backing and wiring of the memory board DRAM /CAS lines would be as described for the 128D in the 256K plans; that is, only pin 16 is lifted and each /CAS wired to that pin "leap frogs" a chip. With or without piggy-backing, if you choose to retain the R.F. shield you will have to cut an opening in it to clear the memory or daughter board components. The most ambitious solution to the dimensional constraints of the low-profile 128 would be to repackage it in a new case with a profile not quite so low. A metal case would eliminate the need for an R.F. shield.

SOFTWARE CONSIDERATIONS

Program 1 performs a VERY simple test of all eight RAM blocks. When RUN, the first two columns display the bank numbers and the ram blocks being accessed. The last column should display the values 0-7. Obviously, since only one memory location is tested in each block, this is not an exhaustive performance test. Note that system IRQs are disabled while program 1 executes. While the operating system "knows" about RAM 2/3, and initializes them upon start-up, it knows nothing about the new RAM 4-7.

A very important part of initialization is the copying of identical routines into the same locations ($FF05-$FF44) in all RAM blocks. These routines handle IRQs, NMIs, and RESET by saving the 8502 registers, the current configuration, switching in system ROM and JMPing to the appropriate handler. Code is not pulled out from under the processor when these routines change banks because execution continues in the ROM version of the routine. Fortunately, initializing RAM 4-7 is simple enough to even be accomplished in Basic (program 2). The ML initialization (program 3) gets around the problem of code being switched out from under the microprocessor by temporarily enlarging common memory to 16K. Since the ML is relocatable, it can be BSAVEd and BOOTed at any convenient address in BANK 15.

As mentioned earlier, in some cases it may be necessary to use AND and OR when altering bits in the RCR. The four-Doodle program supplied with the 256K plans does this when changing bits 6/7 to display bit maps in the four blocks. The RCR is first PEEKed, and the value is then ANDed with %00111111 (63/$3F). This clears bits 6/7 and preserves bits 0-5 and thus does not change the amount or location of common memory or the presumed SuperBank (now "mode") bits. The result is then ORed with 0, 64, 128 or 192 to allow the VIC to display RAM 0, 1, 2 or 3. This lets the program run on a 256K-modified 128 or in either 256K "group" of a 512K-modified 128. The "Globe" and "Pound" adaptations also run in either 256K group. (See "TRANSFER" program.)

Program 4, "Super 8", changes bits 4/5 of $D506 before each picture is BLOADed to the "extended" RAMs 2/3. The eight pictures are rapidly displayed by changing bits 4/5 as well as 6/7 to display the eight bit-maps. Because this program uses all eight blocks, it runs only in normal mode, so AND and OR are not used in the POKEs to 54534. Programs 2 or 3 MUST be run before running Super 8.

Like the four-Doodle program, Super 8 includes TRAP and a target line to restore the RCR to normal when the program ends or is halted by an error or by pressing the STOP key. (STOP-RESTORE does not return the RCR to its default contents!) During program development, the TRAP target routine should include PRINTing of the error variables EL and ERR$ to display any errors. Do this AFTER the RCR is returned to normal -- normal being 1K of common memory at the low end, VIC displaying RAM 0, and bits 4/5 whatever they were before the program was run. (%00??0100)

TRANSFER ROUTINE

In its current state, program 5 provides a crude form of "task-switching". This ML program could conceivably form the basis of a dual-tasking mode. It begins by testing $D506 (in Bank 15) for #$04, the RCR's normal contents and, therefore, normal mode. Satisfied that the computer is in normal mode, the routine switches to mode "B" and copies part of itself to RAM 4 impersonating RAM 2. It then performs a nearly complete reset, with RAM 4-7 acting as RAM 0-3. (The MMU initialization is skipped, since that would put the machine back where it started from, normal mode). If a bootable disk is in the drive, it will boot. However, GEOS128 crashes in the alternate environment, undoubtably because it alters $D506 without preserving the conditions of bits 4/5. The 8502/Z80 transfer code is missing from $FFD0 of the alternate RAM 0, but even when it's present, CP/M also crashes in the second 256K group, probably for the same reason GEOS does. (Both CP/M and GEOS still function in normal mode, however). Speedscript 128 functions in the second 256K, and so does Spectrum, which means that BASIC 8 does, too. Anything that doesn't mess with $D506 should work properly. Any program that alters the RCR without regard for bits 4/5, always making them %00, will work only in normal mode.

The Basic loader will install the reset-and-transfer routine at an address of your choice. When installed at address SA, BANK15: SYS SA resets to alternate mode. In both environments, BANK15: SYS SA + 100 will allow you to go back and forth from one 256K group to the other. Each time the routine at SA + 100 is called, the processor registers and the RCR contents are saved within the routine. (The value in the Configuration Register at $FF00 is NOT saved, so the routine should be called only in Bank 15). The RCR is then changed to switch in the "other" set of 256K and new values are put in the processor registers. This process is repeated each time the current 256K group is exited. If you lose track of which group you're in, X = PEEK(54534)AND48 (in Bank 15) will give 0 in normal and 48 in alternate.

To use the DOS Shell in the second 256K, insert the disk in the drive AFTER the reset to alternate mode is completed. Then BLOAD "DOS SHELL",B0. Enter the machine language monitor and change the LDA #$04 at $3227 to LDA #$34. Exit, type KEY1, "BANK12:SYS6656" + CHR$(13). Press <return>, then press <f1>. No doubt other programs can just as easily be made to work in alternate mode. Ideally, any program that stores values in $D506 should do so in a way that doesn't disturb bits that are not to be altered. Again, this can be accomplished through the use of AND and OR, in Basic or machine language.

There are many complications to implementing true dual-tasking. The transfer operation would have to occur on each interrupt. Since this would give each environment half as much processing time, 2 MHz (FAST) mode would have to be used to get an effective 1 MHz. This would confine dual-tasking to 80 columns, which is

just as well, since there is only one Color Memory for the VIC to use. (It might be possible to use the "other" Color Memory block normally used for multi-color bit-maps. But the display would alternate between two text screens, probably causing both 40 column screens to be unreadable. Using non-overlapping windows in each 256K group would work in 80 columns, but not in 40). A means must be found to keep disk and other I/O operations from "colliding". Which program receives keyboard input? When STOP-RESTORE is pressed, which program stops? How are "wedges" and even normal interrupt activities affected by the IRQ-driven transfer? These are the sort of puzzles certain masochistic types take great delight in trying to solve.

C64 MODE

In C64 mode this modification can provide as much as 194,560 bytes of RAM but, for that much RAM to be available, 64 mode must be entered in a non-standard way.

Although the MMU documentation states that "there are no preconditions ... to force a particular memory alignment in C64 mode", I have not found that to be entirely true. It appears that some portion of RAM 0 must be present. That is, you can not GO64 and have ALL of any RAM block other than RAM 0. And, apparently, that required portion of RAM 0 must be at the bottom of memory. (I could be mistaken, though).

In 512K normal mode, use the machine language monitor to enter this code at $8000 in TRUE RAM 2 (omit comments):

```
28000 sei
28001 lda #$8e   ; RAM 2, I/O and kernal
28003 sta $ff00
28006 lda #$07   ; display RAM 0, 16K of
28008 sta $d506  ; com. mem. at low end
2800b jmp $ff4d  ; 64 mode
```

Exit to Basic, then load and run the 512K TEST program to store the values 0-7 at location 49152 in all eight RAM blocks. Then type BANK2:SYS32768 <return>. When the 40 column C64 screen appears, ?PEEK(49152) will display "2". RAM 0 is present from address 0 to 16383. Above that, RAM 2, 4 or 6 can be switched in by POKEing 0, 16, or 32 to 54534. Although the MMUs disappear in 64 mode, latch Z7 remains, and we can still alter bits 4/5. The fourth value, making both bits %1 by POKEing 48 in 54534, again gives us RAM 4, but ALL of it. The screen turns into garbage because the lowest 16K of RAM 0 was switched out. (This would be RAM 4 as "alternate" RAM 0 in 128 mode). Without preparation, i.e. ML routines, the only way to recover from this is by a hardware RESET.

If we had stored #$84 in $D506 while in 128 mode, in C64 mode VIC would display RAM 2 and RAM 0 would exist from $0000 to $03ff (1K of common memory). Switching Banks would switch in a new screen as well as new RAM from $0400 to $FFFF. This could only be done from an ML routine in the cassette buffer or some other location below $0400. Switching Banks would also switch out any ML in the $C000-$CFFF range. A further complication is that, because the MMUs disappear in 64 mode, the contents of latch Z7 cannot be read. (In 128 mode, bits 4/5 of the RCR retain whatever is written to them, and we can read the RCR). One solution to this difficulty is to always return to some known "base" configuration.

Of course, ML routines that use techniques like the TRANSFER routine (which uses techniques borrowed from the routines in page $FF,) could be stored in RAM while the computer is in 128 mode. These could give a means around the "vanishing RAM" problem. But since using the added RAM in 64 mode is going to be VERY tricky, we might just ignore the added RAM, always GO64 the normal ways, or disable the modification in 64 mode. If you've installed the disable SWitch, which is recommended, just flip it to the "un-mod" position.* The disabling could be automated by replacing capacitor C1 with a diode to MS3, in the same direction as d1. This would keep latch Z7 cleared and unalterable in 64 mode. With either C1 or a second diode, GO64 DOES NOT WORK INALTERNATE Reset while holding down the C= key ALWAYS works.

* So far, I've had to do this with only one program, an old version of Renegade, a disk backup utility, which is now called Maverick.

Editor's note: Richard can upgrade your "flat" C-128 or C-128D for you. The best way to contact him is to write to him and include a SELF ADDRESSED STAMPED ENVELOPE along with your night telephone number. The cost for the 256K upgrade is $80 plus parts and shipping both ways. The cost for the 512K upgrade is $95 plus parts and shipping both ways.

Write to Richard before sending him your computer so he can schedule your upgrade and give you the latest pricing information for the chips and hardware needed to do the upgrades. I am very happy with my new C-512D!

Richard Curcio
22 Seventh Ave
Brooklyn, NY 11217

512K PROGRAMS

PROGRAM 1 : 512K.PROG1

```
lk 100 rem *** test ram 0-7 ***
ki 110 :
en 120 bank15:poke53274,0:   rem disable vic irqs
mo 130 x=0
id 140 fori=0to7:ifi>3thenx=2
mp 150 ifi=4ori=5thenbank15:poke54534,20:rem banks 4/5 as 2/3
bb 160 ifi=6ori=7thenbank15:poke54534,36:rem banks 6/7 as 2/3
eb 170 bankiand3orx:poke49152,i
dg 180 next
ci 190 bank15:poke54534,4:   rem normal
be 200 x=0
mj 210 fori=0to7:ifi>3thenx=2
pn 220 ifi=4ori=5thenbank15:poke54534,20
lp 230 ifi=6ori=7thenbank15:poke54534,36
oj 240 print"bank"iand3orx;
ni 250 bankiand3orx:print"ram"i;peek(49152)
ih 260 next
pc 270 bank15:poke54534,4:  rem back to normal
ko 280 poke53274,241     :  rem enable irqs
kd 290 end
```

PROGRAM 2 : 512K.PROG2

```
ik 100 rem *** initialize ram 4-7 ***
ki 110 :
be 120 bank15:ifpeek(54534)<>4thenprint"wrong mode!":end
jo 130 poke53274,0          :rem disable vic irqs
dg 140 fori=65285 to 65348 :rem $ff05-ff44
ao 150 gosub230:next
dm 160 fori=65530 to 65535 :rem $fffa-ffff
cc 170 gosub230:next
oo 180 :
ce 190 bank15:poke53274,241:rem enable irqs
ji 200 pokedec("d506"),4    :rem normal
fc 210 end
bg 220 :
ee 230 bank15:x=peek(i)     :rem get data from rom
jl 240 pokedec("d506"),20   :rem ram 4/5 = 2/3
ea 250 gosub310
go 260 bank15
ge 270 pokedec("d506"),36   :rem ram 6/7 = 2/3
fp 280 gosub310
lb 290 return
gh 300 :
fh 310 bank3:pokei,x
fo 320 bank2:pokei,x
nj 330 return
```

PROGRAM 3 : 512K.PROG3

```
ic 100 rem *** initialize ram 4-7 in m.l. ***
ki 110 :
lo 120 sys4000
ln 130 ;
kd 140 ;buddy 128
nb 150 ;
gl 160 .mem
of 170 ;
gd 180 *= $1300
pj 190 ;
ad 200 ; call in bank 15
an 210 ;
mo 220 :        php
ck 230 :        sei             ; no interruptions
lh 240 :        lda $d506
ag 250 :        cmp #$04        ; normal"?
ej 260 :        bne msg         ; no...
mf 270 :        pha
jg 280 :        lda #$27        ; 16k of common mem.
hf 290 model    sta $d506       ; mode 'c' (6/7=2/3)
op 300 :        lda #$80        ; ram 2, rom & i/o
km 310 cnf1     sta $ff00
fl 320 :        ldx #$3f
kc 330 loop1    lda $ff05,x     ; irq, nmi & reset
lp 340 :        sta $ff05,x     ; routines
fa 350 :        dex
aj 360 :        bpl loop1
pa 370 :        ldx #$05
ij 380 loop2    lda $fffa,x     ; "hard" vectors
kc 390 :        sta $fffa,x
ic 400 :        dex
eb 410 :        bpl loop2
mb 420 :        lda #$c0        ; ram 3, rom & i/o
ji 430 :        cmp $ff00       ; is it"?
oa 440 :        bne cnf1        ; no -- make it so
jg 450 :        lda #$17        ; mode 'b'
nj 460 :        cmp $d506
lb 470 :        bne model       ; repeat for ram 4/5
bm 480 ;
oe 490 :        lda #$00
pk 500 :        sta $ff00       ; bank 15
mb 510 :        pla
do 520 :        sta $d506       ; 'normal' mode
fg 530 :        plp             ; status
ee 540 :        rts
gd 550 ;
jg 560 msg      plp
kn 570 :        jsr $ff7d       ; primm
ph 580 .byte    $0d             ; "return"
jo 590 .asc     "wrong mode!"
he 600 .byte    $0d,0
ik 610 :        rts
jm 620 .end
```

512K PROGRAMS

PROGRAM 3 : 512K.INT.LDR

```
kk 100 rem *** init 4-7 in ml ***
gf 110 bank15:sa=4864:rem relocating
jg 120 fori=0to85:readd:pokesa+i,d:next
dl 130 rem use bank15:sys sa
oj 140 data   8,120,173,  6,213,201,  4,208
mc 150 data  58, 72,169, 39,141,  6,213,169
ki 160 data 128,141,  0,255,162, 63,189,  5
ek 170 data 255,157,  5,255,202, 16,247,162
pc 180 data   5,189,250,255,157,250,255,202
gl 190 data  16,247,169,192,205,  0,255,208
ii 200 data 224,169, 23,205,  6,213,208,212
bg 210 data 169,  0,141,  0,255,104,141,  6
mo 220 data 213, 40, 96, 40, 32,125,255, 13
pm 230 data  87, 82, 79, 78, 71, 32, 77, 79
pp 240 data  68, 69, 33, 13,  0, 96
```

PROGRAM 4 : 512K.SUPER 8

```
ic 100 rem *********    super 8    *********
bg 110 rem *    press any key to stop      *
fb 120 rem *    re-start with "run 190"    *
if 130 rem **********************************
mg 140 :
ke 150 trap270:gosub280:bank15:graphic1
hc 160 fori=0to7:poke54534,z(i)
gd 170 bload(n$(i)),b(b(i)),p7168
bo 180 next:poke54534,4
dm 190 trap270:gosub280:rem re-read arrays for re-start
mk 200 bank15:graphic1:x=0
om 210 t=199:rem display duration
lh 220 ifx>7thenx=0
cn 230 poke54534,z(x)
ih 240 fori=0tot:next:rem time delay
hf 250 geta$:ifa$<>""then270
ld 260 x=x+1:goto220
ak 270 bank15:poke54534,4:graphic0:end
kf 280 restore
od 290 fori=0to7:readn$(i),b(i),z(i):next
ll 300 return
hb 310 :
ce 320 rem   name, bank, rcr value
mk 330 data "ddmiddle earth",0,4,"ddopbox1",1,68
da 340 data "ddfront pic",2,132,"ddbackcover",3,196
mn 350 data "ddgraphic blocks",2,148,"ddplan",3,212
aa 360 data "ddspiral",2,164,"ddletters",3,228
kn 370 :
fb 380 rem  some of these pix are art studio
nn 390 rem  converted to doodle
ll 400 rem  you can use any doodle file
md 410 rem  in place of the above doodles
```

PROGRAM 5 : 512K.TRSF.BAS

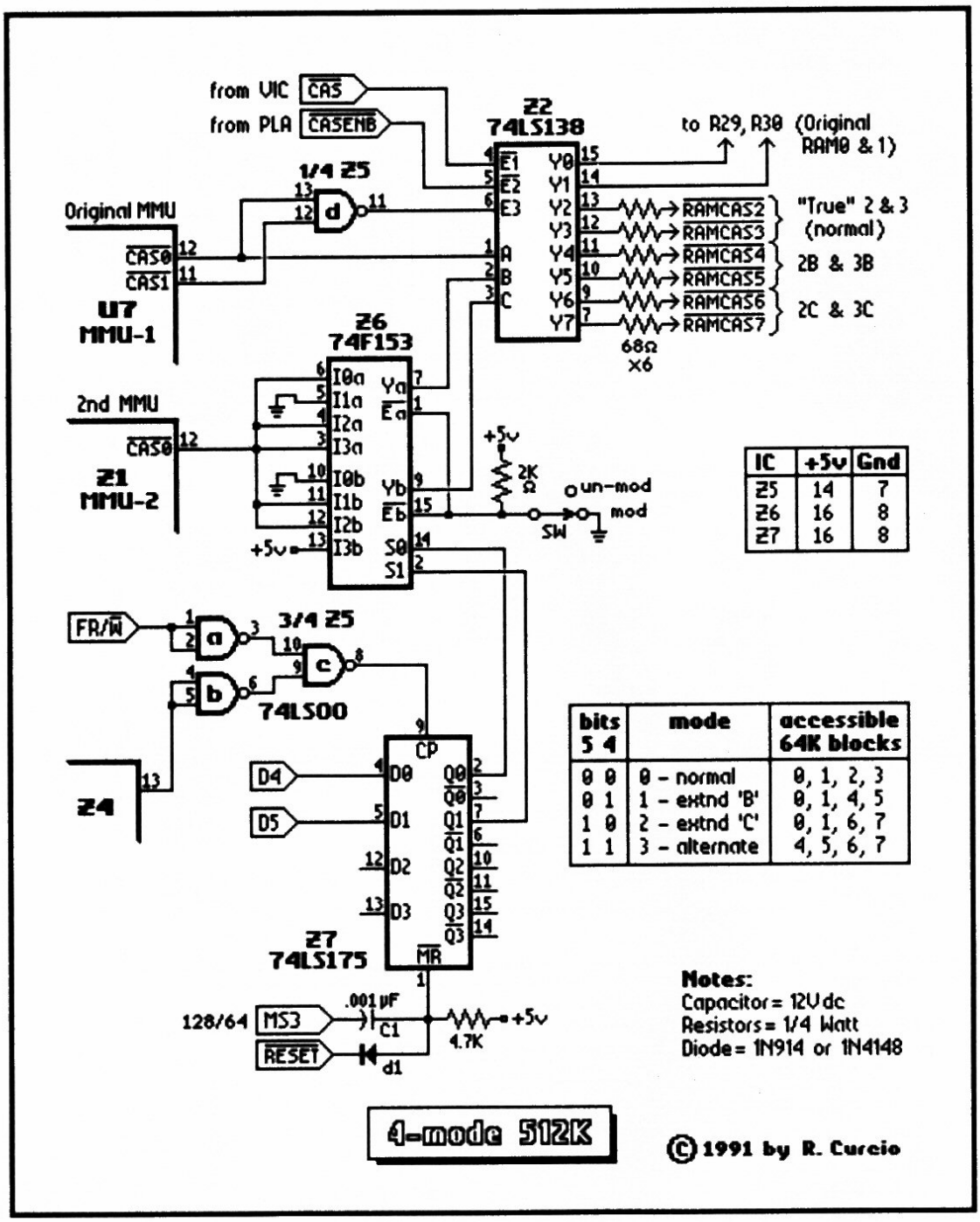
```
bi 100 rem *** transfer loader ***
mm 110 bank15:sa=5120:rem relocating
lg 120 fori=0to149:readd:pokesa+i,d:next
ln 130 ad=sa+96:gosub420
ia 140 pokesa+32,l:pokesa+34,h
ck 150 fori=0to5:readaa,dd:ad=sa+dd:gosub420
lg 160 pokesa+aa,l:pokesa+aa+1,h:next
fe 170 print"reset    = bank15:sys"sa
di 180 print"transfer = bank15:sys"sa+100
do 190 end
pm 200 data 173,  6,213,201,  4,240, 18, 32
ld 210 data 125,255, 13, 87, 82, 79, 78, 71
jo 220 data  32, 77, 79, 68, 69, 33, 13,  0
ak 230 data  96,120,  9, 20,141,  6,213,169
em 240 data  96,160, 20,133,206,132,207,169
je 250 data 206,141,185,  2,160, 53,177,206
eb 260 data 162,  2, 32,119,255,136, 16,246
ak 270 data 160, 30,169, 41,162,  2, 32,119
dj 280 data 255,200,169,207,162,  2, 32,119
nj 290 data 255,200,200,200,200,200,169, 52
pn 300 data 162,  2, 32,119,255,216,162,255
lj 310 data 154,169,  0,160, 52,140,  6,213
co 320 data  76, 21,224,234,234,  8,120,141
ap 330 data 147, 20,142,145, 20,140,143, 20
kg 340 data 104,141,140, 20,186,142,137, 20
ee 350 data 173,  6,213,141,132, 20,  9, 48
bc 360 data 141,  6,213,169,  4,141,  6,213
ei 370 data 162,246,154,169, 48, 72,160,  0
ab 380 data 162,  0,169,  0, 40, 96
ga 390 rem ** adjustments **
ik 400 data 104,147,107,145,110,143,114,140
cf 410 data 118,137,124,132
hj 420 h=ad/256:l=ad-int(ad/256)*256:return
```

PROGRAM 5 : 512K.TRSF.SRC

```
jf 1000 rem ***  transfer control from   ***
ah 1010 rem      normal to alternate mode
dj 1020 :
fa 1030 sys4000
ep 1040 ;
df 1050 ;buddy 128
gd 1060 ;
pn 1070 .mem
hh 1080 ;
pk 1090 *= $1400
il 1100 ;
jf 1110 ;
lg 1120 ; reset to alternate
kj 1130 ;
ef 1140 areset  lda $d506     ; check mode
oa 1150 :       cmp #$04
cb 1160 :       beq copy
ah 1170 :       jsr $ff7d     ;primm
```

PROGRAM 5 : 512K.TRSF.SRC - continued

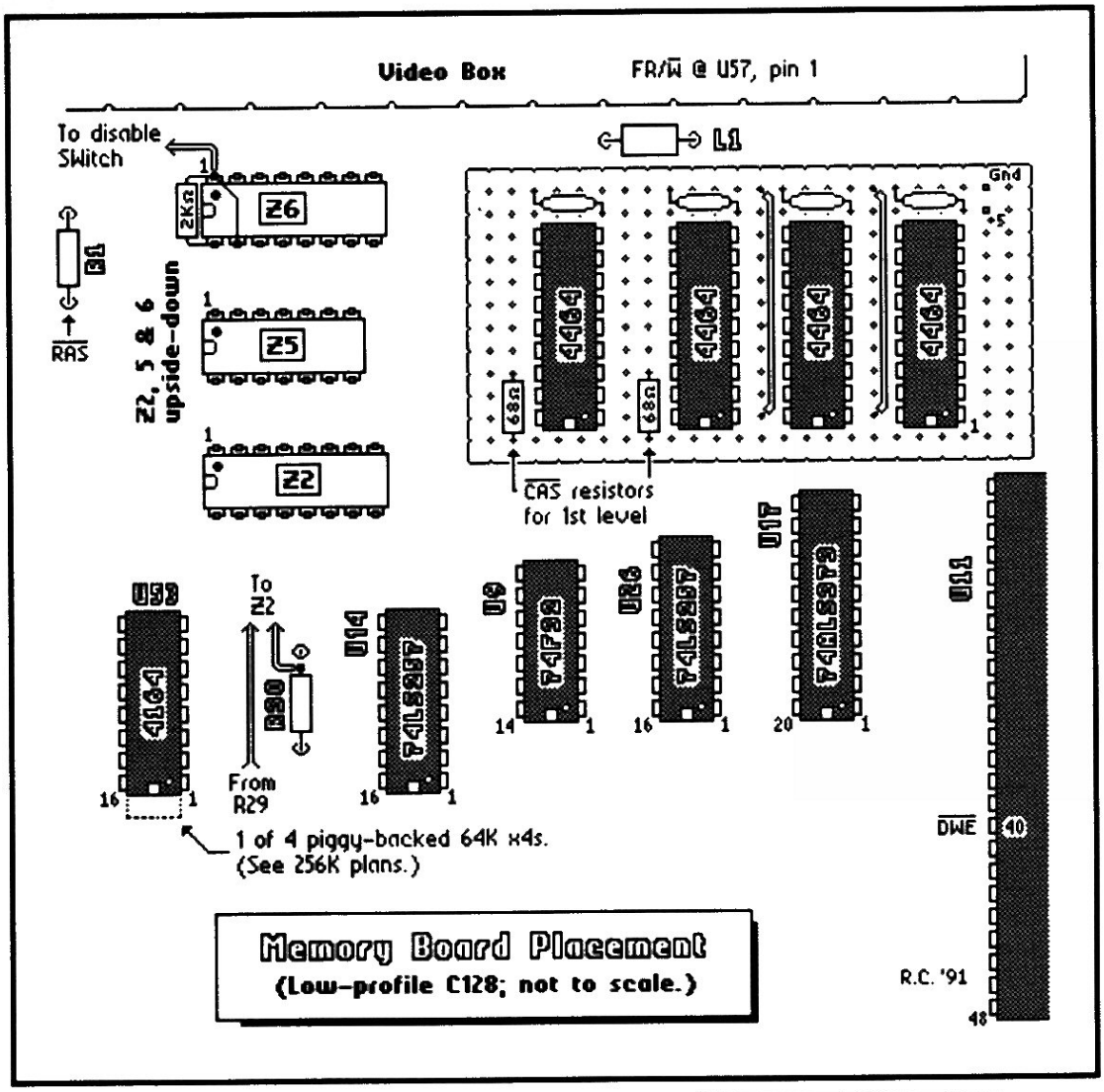
```
eo 1180 .byte   $0d
pi 1190 .asc    "wrong mode!"
mo 1200 .byte   $0d,0
oe 1210 :       rts
ad 1220 ;
na 1230 copy    sei             ; no interruptions
li 1240 :       ora #$14        ; mode 'b' makes
il 1250 :       sta $d506       ; ram 4 = ram 2
ik 1260 :       lda #<switch
oi 1270 :       ldy #>switch
ig 1280 :       sta $ce         ; set-up pointer
oc 1290 :       sty $cf         ; for indsta
dp 1300 :       lda #$ce
eg 1310 :       sta $02b9
gi 1320 ;
jc 1330 ; copy instructions beginning at "switch" to
ne 1340 ; same locations in ram 4 impersonating ram 2
ig 1350 ;
jm 1360 :       ldy #end-switch
mo 1370 cloop   lda ($ce),y
dl 1380 :       ldx #$02        ; bank 2
ho 1390 :       jsr $ff77       ; indsta
hc 1400 :       dey
dm 1410 :       bpl cloop
mm 1420 ;
if 1430 ; make indicated changes
oa 1440 ;
jh 1450 :       ldy #srcr-switch
dc 1460 :       lda #$29        ; "and"
ci 1470 :       ldx #$02
mf 1480 :       jsr $ff77
pb 1490 :       iny
ap 1500 :       lda #$cf
fa 1510 :       ldx #$02
on 1520 :       jsr $ff77
bj 1530 :       iny
ce 1540 :       iny
co 1550 :       iny
di 1560 :       iny
ec 1570 :       iny
ga 1580 :       lda #$34
kb 1590 :       ldx #$02
do 1600 :       jsr $ff77
il 1610 ;
gn 1620 :       cld             ; begin reset sequence
ad 1630 :       ldx #$ff
dc 1640 :       txs             ; stack pntr
hb 1650 :       lda #$00
id 1660 :       ldy #$34        ; altn mode
pb 1670 :       sty $d506
nb 1680 ;
ba 1690 ; the next instruction is never
ei 1700 ; executed in "true" ram 0.
op 1710 ;
jo 1720 switch  jmp $e015       ; later in reset
ml 1730 :       nop
nf 1740 :       nop
bh 1750 ;
an 1760 ; this routine lives in true ram 0 and true
gf 1770 ; ram 4 (pseudo 2 in mode 'b' or second ram 0
gn 1780 ; in alternate mode) with changes as noted.
hl 1790 ; permits jumping from normal to alternate.
ek 1800 ;
cg 1810 xfer    php             ; start here
gh 1820 :       sei             ; no interruptions
ce 1830 :       sta atemp+1
pn 1840 :       stx xtemp+1
ba 1850 :       sty ytemp+1
bm 1860 :       pla             ; retrieve status
nl 1870 :       sta srtemp+1    ; store it
kc 1880 :       tsx             ; stack pointer
om 1890 :       stx pntemp+1
cf 1900 :       lda $d506       ; get ram config reg.
ab 1910 :       sta crtemp+1    ; save for return
ci 1920 srcr    ora #$30        ; change bits 4/5
ed 1930 ;                         (use 'and #$cf'
op 1940 ;                         at same point
ph 1950 :       sta $d506       ; in ram 4)
ok 1960 ;
ii 1970 ; at this point current 256k group is switched
nm 1980 ; out. execution continues at same point in
np 1990 ; "other" 256k group
bc 2000 ;
bb 2010 crtemp  lda #$04        ; first time value
og 2020 :       sta $d506       ; (use #$34 in ram 4)
da 2030 ;
fo 2040 pntemp  ldx #$f6        ; first time
ea 2050 :       txs
jj 2060 srtemp  lda #$30        ; ditto
ne 2070 :       pha
bf 2080 ytemp   ldy #$ff
bg 2090 xtemp   ldx #$ff
fb 2100 atemp   lda #$ff
ee 2110 :       plp
lm 2120 end     rts
ii 2130 .end
```

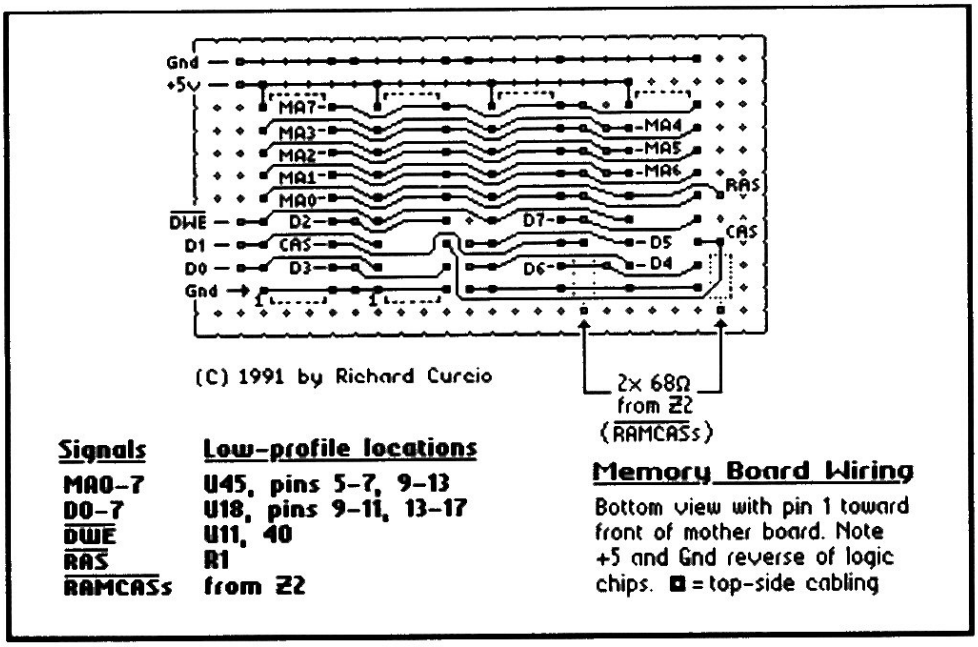


Signal	Location * "Flat"	Location * "D"
MS3	U11, pin 15	-same-
RESET	U6, 40	"
FR/W	U57, 1	U61, 3
D4	U13, 12	-same-
D5	U13, 14	U13, 3

[*] See 256K plans for other signals.

It is possible to build the complete 512K circuit omitting the second 256K. Connect the first four 64K blocks to Z2 pins 12-15, leave pins 7 and 9-11 un-connected, and jumper pin 1 of Z7 to ground. The switch will still enable/disable the mod. When you add the second 256K, remove the ground jumper from pin 1 of Z7.





Signals	Low-profile locations
MA0-7	U45, pins 5-7, 9-13
D0-7	U18, pins 9-11, 13-17
DWE	U11, 40
RAS	R1
RAMCASs	from Z2

Memory Board Wiring

Bottom view with pin 1 toward front of mother board. Note +5 and Gnd reverse of logic chips. ◘ = top-side cabling

This picture to the left is for the 512K upgrade and for C-64 mode !