Auxiliary Bits
(for the +4/C16, B Series, 1541, and 8050)

Elizabeth Deal
Malvern, PA

+4 and C16 Bits

These computers are miracles. What follows are some notes I have which may well be unintelligible to the beginners, but can be of use to someone familiar with other Commodore machines. The User's manual is rather complete, so these are just additional comments:

Character strings are handled differently than in previous machines: there is no such thing as pointers into a program. All strings declared inside a program are copied to RAM (usually hidden under ROM). There are no garbage collection delays. Additionally, new functions can be done with the strings:

1. The first one is assignment statement with MID$ on the left. Yup, you're seeing it correctly. It's now OK to code:

   ```
   MID$(a$,4,2) = "de"
   ```

   This will change whatever was in positions 4 and 5 to be "de". Can you see why strings can't live inside a program? Would be a programming nightmare if they did.

2. INSTR function returns a position of one string within another, so

   ```
   INSTR$("xyz","y")
   ```

   returns 2. You can also specify a starting position for the search. The old code

   ```
   forj = 1tolen(a$):ifx$ = mid$(a$,j,1)thennextj
   ```

   is no longer needed, and the INSTR function is instantaneous! Hurray to Commodore.

Tape is incompatible with other CBM machines. The connector is different, but that's the small part. The timing is different. It seems that the writing goes at about half the speed of the previous units. The code to accomplish tape writing and reading is enormous. Reading is particularly difficult because the TED chip functions differently: there is no such thing as detecting a negative transition - all transitions have to and are being detected in software. The screen is turned off to permit 1.7 mhz operation. Still, it is a slow process.

Tape errors are funny. If you happen to position a tape to the very tippy-end of a program you don't want to load, the computer reports BREAK error (#30) and does not go on to look for the program you do want. Using error trapping (TRAP statement exists in the language!) is the way to go in program mode.

Generally, error numbers when used with tape are wrong. You may get a DEVICE NOT PRESENT ERROR, when you think it should be a FILE NOT FOUND ERROR. You invariably get BREAK ERROR when the end-of-tape header has been read in. This would be only a cosmetic nuisance, were it not for the fact that a STOP-key also causes a BREAK error. It's hard to tell one from another

There is lots of RAM in the machine, and one tends to play a lot of hide-and-seek games in finding things. Some clues:

1. Page 4 contains various indirect routines that permit taking bytes from ROM or RAM. These routines are used by BASIC.
2. In page 7, specifically at $7d7 is an equivalent routine for use by the machine code monitor.
3. BASIC PEEK returns a byte from RAM. Machine Language Monitor returns a byte from ROM. MLM normally saves only RAM, you can't save ROM. BASIC SAVE normally saves RAM. LOAD loads bytes into RAM, as you'd expect. Sometimes you may wish to change the defaults. It can be done:

(a) To PEEK ROM from BASIC: modify a routine in page 4 (at $0494) to ignore the store instruction:

```
poke1176,44 : now peek ROM : poke1176,141
```

This is fairly safe, so long as you DO NOT WORK ANY STRINGS BETWEEN THE TWO POKE1176 INSTRUCTIONS. This, for instance, is the only way you can get at the character generator ROM from BASIC, as far as I can tell.

(b) To peek/save ROM from the MLM, set bit 7 in the byte at $7f8. Incidentally, the monitor has a nice feature - ">7f8 80" is all you need to type in, the ">" sets bytes and displays just one line of memory.

The MONITOR is nice, it's almost like SUPERMON. But there is a serious bug - they chopped the TRANSFER command: T with overlapping addresses does not work in one direction - when the destination is higher in memory than the source. The bytes just write one over another and you lose all your work. A cure: first Transfer to another, non-overlapping area, then do a second transfer to where you wanted to go in the first place.

Colour memory, as in the C64, contains the colour codes for the 1000 screen bytes. One difference, bit 7 is the flashing bit. Funny things happen when you load the C64 colour map into, what's now called, screen attributes map in the Plus 4. You get flashing for nothing.

To change the colour attributes from the C64 POKEs into the COLOR statements, you'll need to add one to each value, as the Plus 4 colour numbers are from 1 to 16. POKE values are still 0–15, but there is little reason to use them.

The keyboard is a delight. Perhaps a bit too soft, but easy to use. The ESC sequences are a joy to use. There is even a pause–all–output key: two keys actually – CTL–S, with any other key restarting the output. There is only one problem: if you use CTL–S during a program run, and use a subsequent GET statement, the S will appear to the GET statement as a real input – I think it's a bug – the keyboard buffer isn't cleared, so you'll have to do it yourself.

Programmable function keys are useful. Unlike in the B machine, most of them have a carriage return at the end. I don't like this feature, but it can be easily changed. Unfortunately, the keys are active inside a running program. Watch out here: if you use GET, and the user pushes the DIRECTORY key, all the letters in 'DIRECTORY' including the carriage return will be delivered to the GET! If you don't want it to happen, there is a way to disable the function keys: either set them all to null ("") inside a program and redefine at the end, or POKE their lengths to zero.

The default colours and luminances for the sixteen colour keys are in RAM. They are in a table in page 1 at $113. You can change them as you wish. Machine code people who love to POKE the stack (auto–run programs!) will have to stay away from this area.

Some of the structured BASIC statements are splendid. For instance, the DO WHILE construct permits you to code a loop that will never execute (FOR–NEXT loops always run at least once, unless you test and skip around). The interpreter seems to be looking ahead, almost like a compiler: it skips the loop and all the loops inside it. EXIT permits leaving a loop early. LOOP UNTIL tests a condition at the end of a loop. What more can we ask?

The character table is half the size of that in the C64. Reverse characters aren't in ROM, they are software–generated. You may have to take this into account if you convert programs from the C64 to the +4/C16 machines. Pointing the character base address is simple. It does require POKEs, a rare event in this machine: using the BASIC method (above) or the monitor transfer command, move the characters to any RAM. Then tell the TED chip about the move: tell location $ff13 the page number of the start of your character definitions, then clear bit 2 at location $ff12. That's all there is to it. Much simpler than in the C64. Incidentally, it is perfectly all right to speak in semi–hex to the BASIC interpreter, hence

```
POKE DEC("FF12"), DEC("71")
```

will tell the chip the character base is at $7000. What's that 1 doing in $71? No connection, nothing. Nothing is a one. I don't know why.

When you play with non–ROM character set, a nasty thing can happen: an exit from the monitor or any error in BASIC resets things only half way back to normal. So the screen becomes a mess. Several solutions: TRAP all errors in BASIC. Do not leave the monitor (guarantees learning machine code by the total immersion method). Hold STOP and push the little reset button. Define a function key to (blindly) type the reverse maneuver to set the character base to the default again. Enjoy the crazy screen sight and push some keys while you do so – it's actually an interesting display.

It is possible to move the screen memory anyplace in RAM. The TED chip needs to be told of the move, of course. $FF14 register is the place to use. However, I know of no way to print on the screen when it's not in the standard location. You can POKE it, you can flip it, you can do all sorts of things with the relocated screen, but no printing. The print command ($FFD2) delivers bytes to the default location and only there. It should be possible, if you must print on a relocated screen, to reroute output to your own routine (page 3 vectors) but I doubt that it's worth the trouble.

The GRAPHIC split screen always splits five lines from the bottom. The bottom five lines are in the text mode, the top is bit–mapped. The constant which controls the split raster line is coded in ROM, hence a bit rough to change. However, there is a link in the interrupt–service code which does permit you to modify the place of the split screen, if you must do it.

Disabling the STOP key is a favourite pastime of many people. It's quite easy on the Plus 4 computer – use a TRAP statement and trap error #30 to resume execution. I know, however, of one situation where the STOP cannot be TRAPped. That is in I/O. Tape LOAD illustrates it quite well, as things are slow: the STOP can be TRAPped after the message "LOADING" would appear, not before. While the computer is searching for a header, it uses another way to test the STOP. It looks directly at the keyboard register in the TED chip, and it never tells BASIC about it. The same is probably true with the serial disk, but it is a bit harder to catch, as things happen faster. A moral: to disable a STOP during I/O use a little machine code, especially if your program uses tape. The whole exercise is almost pointless anyway, as the little reset button lets anyone in. I like that.

B–128, 1541, and 8050 Bits

I wish Commodore would reconsider their decision to drop the B–machines. B–128 is a terrific machine. Sure it's hard to program, but it's fun. It has superb BASIC, superb keyboard, 2mhz clock, it's fast and pleasant to use!

There are an assortment of curiosities about the machine itself and some disk drives:

Happy news: On the B128 the files close themselves! When an error condition causes a disk file to remain open, editing a program line makes the disk whirr a bit and a file gets closed. It's incomplete, but it's not a • file anymore. Clever and useful – if you keep the drive door down, of course.

There is a RESTORE <line number> command in BASIC, just as in +4.

BLOAD "file name" drive,unit,bank,address loads program files and does not cause BASIC to run from the beginning. A splendid feature.