Memory Management

The processor's point of view

The Commodore 64 has access to more memory than its processor can directly handle. This is possible by banking the memory. There are five user configurable inputs that affect the banking. Three of them can be controlled by program, and the rest two serve as control lines on the memory expansion port.

The 6510 MPU has an integrated I/O port with six I/O lines. This port is accessed through the memory locations 0 and 1. The location 0 is the Data Direction Register for the Peripheral data Register, which is mapped to the other location. When a bit in the DDR is set, the corresponding PR bit controls the state of a corresponding Peripheral line as an output. When it is clear, the state of the Peripheral line is reflected by the Peripheral register. The Peripheral lines are numbered from 0 to 5, and they are mapped to the DDR and PR bits 0 - 5, respectively. The 8502 processor, which is used in the Commodore 128, has seven Peripheral lines in its I/O port. The pin P6 is connected to the ASC/CC key (Caps lock in English versions).

```
The I/O lines have the following functions:

   Direction  Line  Function
   ---------  ----  --------
      out      P5   Cassette motor control. (0 = motor spins)
      in       P4   Cassette sense. (0 = PLAY button depressed)
      out      P3   Cassette write data.
      out      P2   CHAREN
      out      P1   HIRAM
      out      P0   LORAM
```

The default value of the DDR register is $2F, so all lines except Cassette sense are outputs. The default PR value is $37 (Datassette motor stopped, and all three memory management lines high). If you turn any memory management line to input, the external pull-up resistors make it to look like it is outputting logical “1”. This is actually why the computer always switches the ROMs in upon startup: Pulling the -RESET line low resets all Peripheral lines to inputs, thus setting all three processor-driven memory management lines to logical “1” level.

The two remaining memory management lines are -EXROM and -GAME on the cartridge port. Each line has a pull-up resistor, so the lines are “1” by default.

Even though the memory banking has been implemented with a 82S100 Programmable _Logic_ Array, there is only one control line that seems to behave logically at first sight, the -CHAREN line. It is mostly used to choose between I/O address space and the character generator ROM. The following memory map introduces the oddities of -CHAREN and the other memory management lines. It is based on the memory maps in the Commodore 64 Programmer's Reference Guide, pp. 263 - 267, and some errors and inaccuracies have been corrected.

The leftmost column of the table contains addresses in hexadecimal notation. The columns aside it introduce all possible memory configurations. The default mode is on the left, and the absolutely most

rarely used Ultimax game console configuration is on the right. (Has anybody ever seen any Ultimax games?) Each memory configuration column has one or more four-digit binary numbers as a title. The bits, from left to right, represent the state of the -LORAM, -HIRAM, -GAME and -EXROM lines, respectively. The bits whose state does not matter are marked with “x”. For instance, when the Ultimax video game configuration is active (the -GAME line is shorted to ground), the -LORAM and -HIRAM lines have no effect.

```
      default                      001x                       Ultimax
       1111   101x   1000   011x   00x0   1110   0100   1100   xx01
10000
-------------------------------------------------------------------------
 F000
       Kernal RAM    RAM    Kernal RAM    Kernal Kernal Kernal ROMH(*
 E000
-------------------------------------------------------------------------
 D000  IO/C   IO/C   IO/RAM IO/C   RAM    IO/C   IO/C   IO/C   I/O
-------------------------------------------------------------------------
 C000  RAM    RAM    RAM    RAM    RAM    RAM    RAM    RAM     -
-------------------------------------------------------------------------
 B000
       BASIC  RAM    RAM    RAM    RAM    BASIC  ROMH   ROMH    -
 A000
-------------------------------------------------------------------------
 9000
       RAM    RAM    RAM    RAM    RAM    ROML   RAM    ROML   ROML(*
 8000
-------------------------------------------------------------------------
 7000

 6000
       RAM    RAM    RAM    RAM    RAM    RAM    RAM    RAM     -
 5000

 4000
-------------------------------------------------------------------------
 3000

 2000  RAM    RAM    RAM    RAM    RAM    RAM    RAM    RAM     -

 1000
-------------------------------------------------------------------------
 0000  RAM    RAM    RAM    RAM    RAM    RAM    RAM    RAM    RAM
-------------------------------------------------------------------------

    *) Internal memory does not respond to write accesses to these
       areas.
```

```
    Legend: Kernal      E000-FFFF       Kernal ROM.

            IO/C        D000-DFFF       I/O address space or Character
                                        generator ROM, selected by
```

```
                                        -CHAREN. If the CHAREN bit is
                                        clear, the character generator
                                        ROM will be selected. If it is
                                        set, the I/O chips are
                                        accessible.

        IO/RAM      D000-DFFF           I/O address space or RAM,
                                        selected by -CHAREN. If the
                                        CHAREN bit is clear, the
                                        character generator ROM will
                                        be selected. If it is set, the
                                        internal RAM is accessible.

        I/O         D000-DFFF           I/O address space.
                                        The -CHAREN line has no effect.

        BASIC       A000-BFFF           BASIC ROM.

        ROMH        A000-BFFF or        External ROM with the -ROMH line
                    E000-FFFF           connected to its -CS line.

        ROML        8000-9FFF           External ROM with the -ROML line
                                        connected to its -CS line.

        RAM         various ranges      Commodore 64's internal RAM.

        -           1000-7FFF and       Open address space.
                    A000-CFFF           The Commodore 64's memory chips
                                        do not detect any memory accesses
                                        to this area except the VIC-II's
                                        DMA and memory refreshes.

NOTE:   Whenever the processor tries to write to any ROM area
        (Kernal, BASIC, CHAROM, ROML, ROMH), the data will get
        "through the ROM" to the C64's internal RAM.

        For this reason, you can easily copy data from ROM to RAM,
        without any bank switching. But implementing external
        memory expansions without DMA is very hard, as you have to
        use a 256 byte window on the I/O1 or I/O2 area, like
        GEORAM, or the Ultimax memory configuration, if you do not
        want the data to be written both to internal and external
        RAM.

        However, this is not true for the Ultimax video game
        configuration. In that mode, the internal RAM ignores all
        memory accesses outside the area $0000-$0FFF, unless they
        are performed by the VIC, and you can write to external
        memory at $1000-$CFFF and $E000-$FFFF, if any, without
        changing the contents of the internal RAM.
```

A note concerning the I/O area

```
The I/O area of the Commodore 64 is divided as follows:

   Address range  Owner
   -------------  -----
     D000-D3FF    MOS 6567/6569 VIC-II Video Interface Controller
     D400-D7FF    MOS 6581 SID Sound Interface Device
     D800-DBFF    Color RAM (only lower nybbles are connected)
     DC00-DCFF    MOS 6526 CIA Complex Interface Adapter #1
     DD00-DDFF    MOS 6526 CIA Complex Interface Adapter #2
     DE00-DEFF    User expansion #1 (-I/O1 on Expansion Port)
     DF00-DFFF    User expansion #2 (-I/O2 on Expansion Port)
```

As you can see, the address ranges for the chips are much larger than required. Because of this, you can access the chips through multiple memory areas. The VIC-II appears in its window every $40 addresses. For instance, the addresses $D040 and $D080 are both mapped to the Sprite 0 X co-ordinate register. The SID has one register selection line less, thus it appears at every $20 bytes. The CIA chips have only 16 registers, so there are 16 copies of each in their memory area.

However, you should not use other addresses than those specified by Commodore. For instance, the Commodore 128 mapped its additional I/O chips to this same memory area, and the SID responds only to the addresses D400-D4FF, also when in C64 mode. And the Commodore 65, or the C64DX, which unfortunately did not make its way to the market, could narrow the memory window reserved for its CSG 4567 VIC-III.

The video chip

The MOS 6567/6569 VIC-II Video Interface Controller has access to only 16 kilobytes at a time. To enable the VIC-II to access the whole 64 kB memory space, the main memory is divided to four banks of 16 kB each. The lines PA0 and PA1 of the second CIA are the inverse of the virtual VIC-II address lines VA14 and VA15, respectively. To select a VIC-II bank other than the default, you must program the CIA lines to output the desired bit pair. For instance, the following code selects the memory area $4000-$7FFF (bank 1) for the video controller:

```
   LDA $DD02 ; Data Direction Register A
   ORA #$03  ; Set pins PA0 and PA1 to outputs
   STA $DD02
   LDA $DD00
   AND #$FC  ; Mask the lowmost bit pair off
   ORA #$02  ; Select VIC-II bank 1 (the inverse of binary 01 is 10)
   STA $DD00
```

Why should you set the pins to outputs? Hardware RESET resets all I/O lines to inputs, and thanks to the CIA's internal pull-up resistors, the inputs actually output logical high voltage level. So, upon -RESET, the video bank 0 is selected automatically, and older Kernals could leave it uninitialized.

Note that the VIC-II always fetches its information from the internal RAM, totally ignoring the memory

configuration lines. There is only one exception to this rule: The character generator ROM. Unless the Ultimax mode is selected, VIC-II “sees” character generator ROM in the memory areas 1000-1FFF and 9000-9FFF. If the Ultimax configuration is active, the VIC-II fetches all data from the internal RAM.

Accessing the memory places 0 and 1

Although the addresses 0 and 1 of the processor are hard-wired to its on-chip I/O port registers, you can access the memory places 0 and 1. The video chip always reads from RAM (or character generator ROM), so you can use it to read also from 0 and 1. Enable the bit-map screen and set the start address of the graphics screen to 0. Now you can see these two memory locations in the upper left corner. Alternatively, you could set the character generator start address to 0, in which case you would see these locations in @ characters (code 0). Or, you can activate a sprite with start address 0. Whichever method you choose, you can read these locations with sprite collision registers. Define a sprite consisting of only one dot, and move it to read the 8 bits of each byte with the sprite to sprite or sprite to background collision registers.

But how can you write to these locations? If you execute the command POKE 53265,59, you will see that the memory place 1 changes its value wildly. If you disable the interrupts (POKE53664,127), it will remain stable. How is this possible? When the processor writes to 0 or 1, it will put the address on the address bus and set the R/-W line to indicate a write cycle, but it does not put the data on the data bus. Thus, it writes “random” data. Of course this data is not truly random. Actually it is something that the video chip left on the bus on its clock half. So, if you want to write a certain value on 0 or 1, you have to make the video chip to read that value just before the store cycle. This requires very accurate timing, but it can be achieved even with a carefully written BASIC program. Just wait the video chip to be in the top or bottom border and the beam to be in the middle of the screen (not in the side borders). At this area, the video chip will always read the last byte of the video bank (by default $3FFF). Now, if you store anything to the I/O port registers 0 or 1 while the video chip is refreshing this screen area, the contents of the memory place $3FFF will be written to the respective memory place (0 or 1). Note that this trick does not work reliably on all computers. You need good RF protection, as the data bus will not be driven at all when the value remains on it.

On the C128 in its 2 MHz mode, you can write to the memory places with an easier kludge. Just make sure that the video chip is not performing the memory refresh (as it would slow down to 1 MHz in that case), and use some instruction that reads from a proper memory location before writing to 0 or 1. Indexed zero-page addressing modes are good for it. I tested this trick with LDX#1 followed by STA $FF,X. As you can read from the instruction timing section of this document, the instruction first reads from $FF (the base address) and then writes to 0. The timing can be done with a simple LDA$D012:CMP$D012:BEQ *-3 loop. But in the C128 mode you can relocate the stack page to zero page, so this trick is not really useful.

You can also read the memory places 0 and 1 much faster than with sprite collisions. Just make the video chip to read from 0 or 1, and then read from non-connected address space ($DE00-$DFFF on a stock C64; also $D700-$D7FF on C128's). Actually, you can produce a complete map of the video timing on your computer by making a loop that reads from open address space, pausing one frame and one cycle in between. And if you are into copy protections, you could write a program on the open address space. Just remember that there must be a byte on the bus for each clock cycle.

These tricks unfortunately do not work reliably on all units. So far I have had the opportunity to try it on three computers, two of which were Commodore 128 DCR's (C128's housed in metal case with a 1571 floppy disk drive, whose controller is integrated on the mother board). One C128DCR drove

some of its data bits too heavily to high state. No wonder, since its housing consisted of some newspapers spread on the floor.