Subject: C128: Writing to $FF03 and $FF04?
Posted by Harry Potter on Tue, 17 Dec 2013 16:54:37 GMT
View Forum Message <> Reply to Message

I am working on programming the Commodore 128 and am wondering: writing to $FF01 switches the MMU to Bank 0 and $FF02 Bank 1. My documentation says that writing to $FF03 switches to Bank 14 and $FF04 Bank 14 with RAM from Bank 1. This makes no sense, because, when I disassembled the Bank 0/1-specific access routines in the first 1k of RAM, they write to $FF01 or $FF02, perform the access, writes to $FF03 or $FF04 then exits. What do these writes *really* do?

Subject: Re: Writing to $FF03 and $FF04?
Posted by Anton Treuenfels on Wed, 18 Dec 2013 06:25:17 GMT
View Forum Message <> Reply to Message

"Harry Potter" <rose.joseph12@yahoo.com> wrote in message
news:c3d362da-109e-453e-8cb0-41b779eec02a@googlegroups.com...
I am working on programming the Commodore 128 and am wondering: writing to $FF01 switches the MMU to Bank 0 and $FF02 Bank 1. My documentation says that writing to $FF03 switches to Bank 14 and $FF04 Bank 14 with RAM from Bank 1. This makes no sense, because, when I disassembled the Bank 0/1-specific access routines in the first 1k of RAM, they write to $FF01 or $FF02, perform the access, writes to $FF03 or $FF04 then exits. What do these writes *really* do?

==========

Harry, Harry, Harry -

You've really got to stop thinking that just because YOU don't understand what's going on that it "makes no sense". The documentation IS correct, the routines DO behave that way, and there IS a reason.

Frankly I think it would be a waste of time to try to explain it to you; you'll have to figure it out on your own. But here's a hint: it all has to do with how to access RAM memory from ROM routines when they both occupy the same address space.

- Anton Treuenfels

Subject: Re: Writing to $FF03 and $FF04?
Posted by epc8 on Wed, 18 Dec 2013 20:50:43 GMT
View Forum Message <> Reply to Message

On Wednesday, December 18, 2013 1:25:17 AM UTC-5, Anton Treuenfels wrote:

> "Harry Potter" <rose.joseph12@yahoo.com> wrote in message
> news:c3d362da-109e-453e-8cb0-41b779eec02a@googlegroups.com...
> I am working on programming the Commodore 128 and am wondering: writing to
> $FF01 switches the MMU to Bank 0 and $FF02 Bank 1. My documentation says
> that writing to $FF03 switches to Bank 14 and $FF04 Bank 14 with RAM from
> Bank 1. This makes no sense, because, when I disassembled the Bank
> 0/1-specific access routines in the first 1k of RAM, they write to $FF01 or
> $FF02, perform the access, writes to $FF03 or $FF04 then exits. What do
> these writes *really* do?
> ==========
> Harry, Harry, Harry -
> You've really got to stop thinking that just because YOU don't understand
> what's going on that it "makes no sense". The documentation IS correct, the
> routines DO behave that way, and there IS a reason.
> Frankly I think it would be a waste of time to try to explain it to you;
> you'll have to figure it out on your own. But here's a hint: it all has to
> do with how to access RAM memory from ROM routines when they both occupy the
> same address space.
> - Anton Treuenfels

The easiest way to handle bank switching is to call code that is located somewhere else. Suppose you are running code in RAM that wants to call a system routine in ROM. One way might be to store the target address of the ROM routine in two z-page locations. Then call a service routine on zpage. That routine uses memory mapped-I/O to switch out your RAM and switch in ROM. After jumping indirect through your two z-page locs (or the equivalent), execution returns to your service routine which disables ROM, enables RAM and then exits back to your code in RAM.

[Hint, a well known 8 bit micro has onboard ROM and 8 (or 7) expansion slots. It does this all day long for I/O and to add software based languages to the computer. The original IBM-PC did something similar. It has cassette BASIC in ROM which was later modified, and in some cases PATCHED by disk BASIC.]

--- e

Subject: Re: Writing to $FF03 and $FF04?
Posted by Harry Potter on Thu, 19 Dec 2013 14:12:18 GMT
View Forum Message <> Reply to Message

On Wednesday, December 18, 2013 3:50:43 PM UTC-5, ep...@juno.com wrote:
> The easiest way to handle bank switching is to call code that is located somewhere else. Suppose you are running code in RAM that wants to call a system routine in ROM. One way might be to store the target address of the ROM routine in two z-page locations. Then call a service routine on zpage. That routine uses memory mapped-I/O to switch out your RAM and switch in ROM. After jumping indirect through your two z-page locs (or the equivalent), execution returns to your service routine which disables ROM, enables RAM and then exits back to your code in RAM.
>
I already did the run-around issue with MemBank128/cc65: it runs your code from Bank 0 and

calls routines in the first 16k of RAM to access the kernel ad I/O. Bank 1 access is achieved through routines copied to the BASIC input buffer. The question here is what Bank 14 actually *is*. I guess I should look it up myself. I should have access to the C128 PRG this weekend.
