The $D000-$DFFF issues of music code/data

After reading the tutorial Richard Bayliss submitted on “Playing music at $A000 - $FFFF “behind” kernal”, I thought I'd make out some important points:

Problem 1

There may be the case where you include a music file that can start at any address BEFORE $D000, and leads to using up memory BETWEEN $D000-$DFFF. For example, a tune stored between $A000-$E757… say, it's a collection of sub-tunes, or a tune with sample data.

When you compile the source code, you will find that the compiled program completely fails to play the music (or even worse, not run at ALL!) Why is that then?

This is simply because $D000-$DFFF is a reserved bank for I/O registers (SID registers, VIC registers, CIA registers and registers to expanded hardware… e.g. second SID chip).

Solution 1

Setting $01 to #$35 does not give you access to $D000-$DFFF RAM, as it is still hidden underneath. To get access to this RAM, you need to set $01 to #$30 instead (* or any value where the first 4 least significant bits are zero). This condition switches off ALL ROM banks and gives you full access to the 64K of RAM.

Problem 2

This, however, leads to another big problem. Most music routines out there write directly to the SID registers when the “play” subroutine is called. So, when you have $01 set to #$30* and call the “play” address (e.g. JSR $1003) you will find that the SID registers are not written and your computer remains silent. This is simply because the SID register data fails to write to I/O registers between $D400-$D418, and instead writes to RAM. This in fact leads to ANOTHER problem in the case of music data stored between $D400-$D419. I won't go into this detail, but I assure you will find that your music syncing/sounds will be messed up.

Solution 2

Luckily, some good coders out there thought up a way to bypass this. This is called “ghost-registers”. Ghost registers act the same way as a double-buffer. All you need to do is replace, for example, “STA $D400” with “STA $4000”… in other words, the SID register data is stored between $4000-$4018,

rather than $D400-$D418. When you have finished calling the play subroutine, set $01 back to #$35/$37, then read the data from the “ghost” registers, and transfer the values to the SID registers, like so:

```
   lda #$30     ; Switch off I/O and ROM banks
   sta $01
   jsr $d003    ; Play address of an example tune stored from $d000-$e000

   lda #$37     ; Switch the I/O bank back on
   sta $01

   ldx #$18     ; Transfer data from ghost-registers to SID-registers
-  lda $4000,x
   sta $d400,x
   dex
   bpl -
```

As far as I'm aware, the newest music editors, like SDI and Goat-Tracker, has this option to set “ghost registers”, saving you time and effort to go through the entire music routine code and replace the SID register addresses with ghost register addresses. Of course, ghost registers is kind of a new thing, therefore older tunes will require you to go through this painfull practise anyway.

Conclusion

It is actually a very stupid idea to compile a tune that has data stored within the address range of the I/O bank. If you really must have a tune stored within $A000-$FFFF for your production/demo, please make sure that important code and/or music data is not stored within the I/O bank range, so that you don't have to go through the problems and solutions I've explained above.

Thanks for reading! by Conrad/Viruz (14th April 2012)

Links of Interest

http://science.webhostinggeeks.com/problemi-sa-d000 On 13th September 2012, a computer science student from the University of Belgrade (Serbia), wrote a Serbo-Croatian translation of this article as an example to help people from Ex-Yugoslavia better understand some very useful information about computer science.