|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Lasse**  C64 Games Programmer C64 Games Programmer https://www.lemon64.com/forum/images/avatars/2856758435b74a666b8545.gif  Joined: 07 Jan 2002 Posts: 3477 Location: Finland | |  |  | | --- | --- | | [Post](https://www.lemon64.com/forum/viewtopic.php?p=675953#675953)Posted: Wed May 06, 2015 11:53 am    Post subject: | [Reply with quote](https://www.lemon64.com/forum/posting.php?mode=quote&p=675953) | |  | | | Running everything in IRQ to keep things synched is a nice starting point for simpler or not very CPU intensive games (where you can guarantee that you will not use more time than the IRQs allow), but consider what do you actually need synched to the display?   Raster transitions and sprite multiplexing: yes  Displaying the next frame: yes  Scrolling: if you use double-buffering, only the color-RAM scroll needs to be synched, but updating the hidden screen RAM buffer doesn't.  Game logic: no   I advocate using a "command flag" to tell IRQs when they should display the next frame. This needs double (or even triple) buffering all the display information, like scrolling registers and sprite register data, so that you can work on the next frame(s) when the IRQs are still displaying the current frame.   Typically the IRQ would clear the command flag when it has made the next frame visible, so that the main program can watch it and avoid going too fast.   While you're waiting for the flag to clear, you can already perform the next frame's non-synched work, like game logic, and scrolling the hidden screen buffer. This allows you to "run ahead" of the displayed frames most of the time, so when a scrolling update causes a frame to take longer, you hopefully still have enough rastertime available to make the next frame update in time and avoid visible hitching.   If you want to use character bullets (like Turrican), the process becomes a little harder, as those need to be drawn & wiped synched to the display to avoid flickering. In that case I'd recommend the following division of work:   - Main program: queue up game frames. These include scrolling registers, sprite registers, char bullet information, and a work flag telling into which direction the screen should be scrolled (if any)  - Low-level IRQ: take frames from the frame queue for displaying, perform scrolling and drawing the char bullets. This would fire up in the bottom of the screen  - Realtime IRQs: perform raster effects, music playback and sprite multiplexing. These may interrupt the low-level IRQ which may still be e.g. scrolling the screen | | |