NMI lock

From Go64!/CW-issue 09/1999

By Wolfram Sang (Ninja/The Dreams - www.the-dreams.de) Final section added by Frantic/HT after some confused discussions on the CSDb forum.

Don't rely on names!

NMI is short for “Non-Maskable Interrupt” what means, you can't disable it. But as we talk about C64, there is of course a way to do so.

Some demo-effects or transmission routines have a very critical timing. Hit RESTORE once and everything is gone. Even if the NMI-vector points to a RTI, so the Interrupt will be quit immediately, it will “cost” 13 cycles, but just one would be far too much. So it can be quite useful to disable the NMI, though it is a little against its purpose. First I am going to talk about the theory, if you are just interested in the result, then copy the corresponding lines from the example source code. To avoid confusion I want to mention before, that IRQ and NMI are both interrupts, where NMI has higher priority. As the IRQ is more often used, some people call it just “interrupt”. Please don't mix it up! I will use exact definitions in this article.

Theory...

First question is, how are interrupts on the C64 generated? Let us examine the “standard” IRQ. The CPU checks its IRQ-line for a LOW-signal. If there is one, an interrupt will be initiated. It has to be acknowledged via registers $D019 (for VIC) or $DC0D (for CIA1), so the IRQ-line goes HIGH again. If you forget that, there will be another interrupt right after the first one, as there is still a LOW-signal! Okay? Good.

Concerning NMI this procedure is a little different. Not the LOW-signal forces this interrupt, but the change from HIGH to LOW. Of course, the NMI-line has to be raised as well, register $DD0D does that job for CIA2, and some hardware logic for the RESTORE-key. And here is the trick: Don't do that, and nothing special will happen. We still have a LOW-signal, but remember, it is the changing from HIGH to LOW which initiates an interrupt. Best thing is, other incoming HIGH-signals (e.g. from RESTORE) will be absorbed. Aforementioned change is not possible anymore, the CPU will never get to know, if an NMI is requested. I guess, this can be called “disabled NMI”, to reenable it, just acknowledge via $DC0D, so do what you have intentionally “forgotten” before.

... and praxis

For a better understanding I'll give you some comments on the example-sourcecode. If you are not familiar with CIA-Timers, I recommend getting some descriptions of them, because it would be too much for this article, if I should explain them here. At the beginning the programm disables IRQ-interrupts and changes the NMI-vector to our own routine. Then Timer A of CIA2 is stopped and

loaded with 0, so after setting it as NMI-source and starting it, a NMI is going to occur. The interrupt routine increases the border-color, which is only for illustration purpose, and then exits the interrupt without reading $DD0D. The interrupt is not acknowledged! The main programm now continuously changes the upper-left corner of the text-screen, again only for illustration. Try pressing RESTORE, which would normally cause a NMI, but nothing will happen, the border-color stays the same. For a counterexample just press SPACE. The Timer A will be cleared as NMI-source (otherwise it would initiate another NMI and disable it again) and NMI-line will be set HIGH by acknowledging through reading $DD0D. As NMI is allowed again, pressing RESTORE will increase border-color.

Well, that's it! Like most good tricks, it is not much, you just have to know, how it works. If you have problems in understading at first, take your time and try it again. It might look complicated, but it is not, honestly.

Source Code

```
    ; 'Disable NMI' by Ninja/The Dreams/TempesT


        SEI             ; disable IRQ
        LDA #LO(NMI)    ;
        STA $0318       ; change NMI vector
        LDA #HI(NMI)    ; to our routine
        STA $0319       ;
        LDA #$00        ; stop Timer A
        STA $DD0E       ;
        STA $DD04       ; set Timer A to 0, after starting
        STA $DD05       ; NMI will occur immediately
        LDA #$81        ;
        STA $DD0D       ; set Timer A as source for NMI
        LDA #$01        ;
        STA $DD0E       ; start Timer A -> NMI

                        ; from here on NMI is disabled
LOOP:
         INC $0400      ; change screen memory, proves computer is alive
        LDA #$10        ; SPACE pressed?
        AND $DC01       ;
        BNE NOSPC       ; if not, branch
        LDA #$01        ; if yes, clear Timer A
        STA $DD0D       ; as NMI source
        LDA $DD0D       ; acknowledge NMI, i.e. enable it

NOSPC:
        JMP LOOP        ; endless loop

NMI:
         INC $D020      ; change border colour, indication for a NMI
        RTI             ; exit interrupt
```

```
                                ; (not acknowledged!)
```

Aren't NMI interrupts blocking IRQ interrupts?

Some of you might have learned that IRQ interrupts are not served when NMI interrupts are being served. …and you are right! Then a reasonable question is: when using the method to block NMI interrupts described above, why aren't IRQ's disabled too if NMI's are disabled? After all, the method relies on trigging an interrupt, but never acknowledging it. The answer is that IRQ interrupts are not disabled just because NMI interrupts are not acknowledged. As soon as the RTI of the NMI interrupt is executed, IRQ interrupts will be served again. The crucial thing to note here is that you should not mix up the execution of an RTI instruction and acknowledging interrupts. You can do one of them, without doing the other. This means that if you want to be able to use IRQ interrupts when NMI interrupts are disabled, you can safely “neglect” (intentionally) to acknowledge the NMI interrupt, as described in the article above, but that you should make sure that you execute an RTI instruction.

To be even more precise IRQs are not allowed until the interrupt mask flag of the CPU is cleared. When an NMI starts, the I flag is set automatically and RTI clears the I flag among other things. If you clear the mask manually (using CLI) before quitting the NMI code with RTI, you can serve IRQs while executing your NMI.