The double (raster) IRQ method

Theory

This method is one of the more easy methods to grasp when it comes to stable timing. The following explanation of the method assumed you've accustomed yourself to the previous articles in this series.

As stated earlier a raster IRQ will trigger at cycle 0 on the chosen scanline, but due to the fact that the CPU must finish the current opcode + that the CPU needs 7 cycles to actually save the state and move the PC to the IRQ-handle we'll end up somewhere on cycle 7-14 (assuming no sprites on this raster line).

So, due to the finish-current-opcode problem we have a jitter of 7 cycles. However the shorted opcode (in time) is a NOP f.e., so wouldn't it be nifty if we knew that the CPU was executing NOPs when the raster IRQ occured? That would me still 7 cycles for IRQ-setup, but only a jitter of 1 cycle. I.e. we would end up on cycle 7 or 8.

A 2 cycle NOPs only gives a 1-cycle jitter because either the CPU has just finished the current opcode and can thus jump immediatly to the IRQ-handle, or it is half way through the NOP and must stall and additional one cycle to finish it.

Now, how to guarantee the CPU to execute NOPs when the IRQ occurs?

It's quite easy, let's configure a raster IRQ at raster line X to pull the CPU out of the main-code. This IRQ will jitter with 7 cycles at most since the main code can be anything. Now, immediatly let's configure the VIC to trigger another raster IRQ at line X+1. We're almost at X+1 but quite not there yet so let's just fill up with NOPs. This would ensure that when the second IRQ occurs we know 100% that the CPU will be executing NOPs and thus give us a jitter of only 1 cycle at line X+1.

Remember that the C64 has two irq pointer locations? One at $fffe in the ROM memory which is due to hardware design and one at $0314 in RAM which is just a memory location the kernal uses to indirect call a user irq handler. We can use both pointer to accomplish the task of irq jitter correction. The first irq, which contains the “nop” list, is called by the $0314 pointer with a cpu port value of $37 or $36. The second irq which just has a jitter of 1 cycle is called by the rom pointer at $fffe with a cpu port value of $35. The trick is to set up these pointers beforehand and within the irq code just toggle the cpu port value to toggle the irq targets. The number of nops can be determined by just removing one after each other until the part crashes. Then add again one nop. This is how it is done in the following code example where $0314 points to “irq0” and $fffe points to “irq”. The three pla just removes the pc and status from the stack which were pushed at the moment the second irq was triggered. However after the second irq ends we want to jumpback to the main program and not somewhere in the middle of the nop list. The inc $d012 in the first irq must come early so we sntill in the rasterline previous to the rastene in which the secondn irq should be triggered. The same applies for the dec $d012 in the second irq for similar reasons. After the three pla some timing opcodes follow. These assure that we are at the end of a rasterline when the lda $d012 is beeing performed. The compare tests, if the load occured in rasterline N or N+1 and wastes a cycle in the first case. As an effect the lines after the branch are executes always at the same rasterbeam position. The irq is stable, now.

```
.PROC irq0
         inc $d012
         lsr $d019
         dec $01
         cli
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
.ENDPROC

.PROC irq
         dec $d012

         pla
         pla
         pla

         lda (0,x)
         lda (0,x)
         lda (0,x)
         lda (0,x)
         nop
         bit 2
         bit 2
         lda $d012
         cmp #$39
         beq fixcycle
fixcycle bit $d020
...
```

The corresponding irq init looks like this:

```
.PROC irqInit
        lda #$7f
```

```
         sta $dc0d
         bit $dc0d
         lda #$81
         sta $d01a
         lda #<irq0
         sta $0314
         lda #>irq0
         sta $0315
         lda #<irq
         sta $fffe
         lda #>irq
         sta $ffff
         lda #$38
         sta $d012
         lda #$36
         sta $01
         lsr $d019
     rts
.ENDPROC

The lsr $d019 at the end of the init is performed to clear the irq request
if for some reason a rasterirq request is rised during init.
```