

How Real Programmers Acknowledge Interrupts

With RMW instructions

```

; beginning of combined raster/timer interrupt routine
LSR $D019      ; clear VIC interrupts, read raster interrupt flag
to C
BCS raster     ; jump if VIC caused an interrupt
...           ; timer interrupt routine

```

Operational diagram of LSR \$D019:

#	data	address	R/W	
1	4E	PC	R	fetch opcode
2	19	PC+1	R	fetch address low
3	D0	PC+2	R	fetch address high
4	xx	\$D019	R	read memory
5	xx	\$D019	W	write the value back, rotate right
6	xx/2	\$D019	W	write the new value back

The 5th cycle acknowledges the interrupt by writing the same value back. If only raster interrupts are used, the 6th cycle has no effect on the VIC. (It might acknowledge also some other interrupts.)

With indexed addressing

```

; acknowledge interrupts to both CIAs
LDX #$10
LDA $DCFD,X

```

Operational diagram of LDA \$DCFD,X:

#	data	address	R/W	description
---	---	---	---	-----

1	BD	PC	R	fetch opcode
2	FD	PC+1	R	fetch address low
3	DC	PC+2	R	fetch address high, add X to address low
4	xx	\$DC0D	R	read from address, fix high byte of address
5	yy	\$DD0D	R	read from right address

```

; acknowledge interrupts to CIA 2
LDX #$10
STA $DDFD,X

```

Operational diagram of STA \$DDFD,X:

#	data	address	R/W	description
1	9D	PC	R	fetch opcode
2	FD	PC+1	R	fetch address low
3	DC	PC+2	R	fetch address high, add X to address low
4	xx	\$DD0D	R	read from address, fix high byte of address
5	ac	\$DE0D	W	write to right address

With branch instructions

```

; acknowledge interrupts to CIA 2
      LDA #$00 ; clear N flag
      JMP $DD0A
DD0A  BPL $DC9D ; branch
DC9D  BRK      ; return

```

You need the following preparations to initialize the CIA registers:

```

      LDA #$91 ; argument of BPL
      STA $DD0B
      LDA #$10 ; BPL
      STA $DD0A
      STA $DD08 ; load the ToD values from the latches
      LDA $DD0B ; freeze the ToD display
      LDA #$7F
      STA $DC0D ; assure that $DC0D is $00

```

Operational diagram of BPL \$DC9D:

#	data	address	R/W	description
1	10	\$DD0A	R	fetch opcode
2	91	\$DD0B	R	fetch argument
3	xx	\$DD0C	R	fetch opcode, add argument to PCL
4	yy	\$DD9D	R	fetch opcode, fix PCH

```
( 5 00 $DC9D R fetch opcode )
```

```
; acknowledge interrupts to CIA 1
```

```
    LSR          ; clear N flag
```

```
    JMP $DCFA
```

```
DCFA BPL $DD0D
```

```
DD0D BRK
```

```
; Again you need to set the ToD registers of CIA 1 and the
; Interrupt Control Register of CIA 2 first.
```

```
Operational diagram of BPL $DD0D:
```

#	data	address	R/W	description
1	10	\$DCFA	R	fetch opcode
2	11	\$DCFB	R	fetch argument
3	xx	\$DCFC	R	fetch opcode, add argument to PCL
4	yy	\$DC0D	R	fetch opcode, fix PCH
(5	00	\$DD0D	R	fetch opcode)

```
; acknowledge interrupts to CIA 2 automagically
```

```
; preparations
```

```
LDA #$7F
```

```
STA $DD0D          ; disable all interrupt sources of CIA2
```

```
LDA $DD0E
```

```
AND #$BE          ; ensure that $DD0C remains constant
```

```
STA $DD0E          ; and stop the timer
```

```
LDA #$FD
```

```
STA $DD0C          ; parameter of BPL
```

```
LDA #$10
```

```
STA $DD0B          ; BPL
```

```
LDA #$40
```

```
STA $DD0A          ; RTI/parameter of LSR
```

```
LDA #$46
```

```
STA $DD09          ; LSR
```

```
STA $DD08          ; load the ToD values from the latches
```

```
LDA $DD0B          ; freeze the ToD display
```

```
LDA #$09
```

```
STA $0318
```

```
LDA #$DD
```

```
STA $0319          ; change NMI vector to $DD09
```

```
LDA #$FF          ; Try changing this instruction's operand
```

```
STA $DD05          ; (see comment below).
```

```
LDA #$FF
```

```
STA $DD04          ; set interrupt frequency to 1/65536 cycles
```

```
LDA $DD0E
```

```
AND #$80
```

```
ORA #$11
```

```
LDX #$81
```

```
STX $DD0D          ; enable timer interrupt
```

```
        STA $DD0E        ; start timer

        LDA #$00         ; To see that the interrupts really occur,
        STA $D011        ; use something like this and see how
LOOP    DEC $D020        ; changing the byte loaded to $DD05 from
        BNE LOOP        ; #$FF to #$0F changes the image.
```

When an NMI occurs, the processor jumps to Kernal code, which jumps to (\$0318), which points to the following routine:

```
DD09    LSR $40          ; clear N flag
        BPL $DD0A        ; Note: $DD0A contains RTI.
```

Operational diagram of BPL \$DD0A:

#	data	address	R/W	description
1	10	\$DD0B	R	fetch opcode
2	11	\$DD0C	R	fetch argument
3	xx	\$DD0D	R	fetch opcode, add argument to PCL
4	40	\$DD0A	R	fetch opcode, (fix PCH)

With RTI

```
; the fastest possible interrupt handler in the 6500 family
; preparations
SEI
LDA $01        ; disable ROM and enable I/O
AND #$FD
ORA #$05
STA $01
LDA #$7F
STA $DD0D      ; disable CIA 2's all interrupt sources
LDA $DD0E
AND #$BE      ; ensure that $DD0C remains constant
STA $DD0E     ; and stop the timer
LDA #$40
STA $DD0C     ; store RTI to $DD0C
LDA #$0C
STA $FFFA
LDA #$DD
STA $FFFFB   ; change NMI vector to $DD0C
LDA #$FF     ; Try changing this instruction's operand
STA $DD05    ; (see comment below).
LDA #$FF
STA $DD04    ; set interrupt frequency to 1/65536 cycles
LDA $DD0E
```

```
        AND #$80
        ORA #$11
        LDX #$81
        STX $DD0D      ; enable timer interrupt
        STA $DD0E      ; start timer

        LDA #$00      ; To see that the interrupts really occur,
        STA $D011     ; use something like this and see how
LOOP    DEC $D020     ; changing the byte loaded to $DD05 from
        BNE LOOP      ; #$FF to #$0F changes the image.
```

When an NMI occurs, the processor jumps to Kernal code, which jumps to (\$0318), which points to the following routine:

```
DD0C    RTI
```

How on earth can this clear the interrupts? Remember, the processor always fetches two successive bytes for each instruction.

A little more practical version of this is redirecting the NMI (or IRQ) to your own routine, whose last instruction is JMP \$DD0C or JMP \$DC0C. If you want to confuse more, change the 0 in the address to a hexadecimal digit different from the one you used when writing the RTI.

Or you can combine the latter two methods:

```
DD09    LSR $xx ; xx is any appropriate BCD value 00-59.
        BPL $DCFC
DCFC    RTI
```

This example acknowledges interrupts to both CIAs.

If you want to confuse the examiners of your code, you can use any of these techniques. Although these examples use no undefined opcodes, they do not necessarily run correctly on CMOS processors. However, the RTI example should run on 65C02 and 65C816, and the latter branch instruction example might work as well.

The RMW instruction method has been used in some demos, others were developed by Marko Mäkelä. His favourite is the automagical RTI method, although it does not have any practical applications, except for some time dependent data decryption routines for very complicated copy protections.

From:

<https://codebase64.org/> - **Codebase 64 wiki**

Permanent link:

https://codebase64.org/doku.php?id=base:how_real_programmers_acknowledge_interrupts

Last update: **2015-04-17 04:32**

