« Assembly Evolution Part 1: Accessing Memory and the strange case of the Intel 4004

Clockslide: How to waste an exact number of clock cycles on the 6502

by Sven Oliver 'SvOlli' Moll; the original German language version has been simultaneously posted on his blog.

This is an article about the 6502 processor about the topic: how to "waste" a number of clock cycles stated in a register, in this case the X register. The principle is simple: you have a number of operations that do close to nothing. The more the code is jumped to at the "front", the more clock cycles are needed to get to the actual code. If the code is jumped to more at the "end", the CPU gets to the code in question more quickly.

This nice theory won't work directly on the 6502, because every instruction takes at least two clock cycles to execute. If you want to get it down to the precision of one cycle, this is getting more difficult. The first half of this trick I found in code of Eckhard Stollberg, who is one of the guys that pionieered homebrew on the Atari 2600 VCS. There, I found some strange bytes:

```
C9 C9 C9 C9 C9 C9 C9 C9 C9 C9 C9 C5 EA
```

The disassembly looks like this:

```
; CODE1
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP $EA  ; 3
```

To run through the code, you'll need 15 clock cycles, and nothing changes except for some state registers. If the code is called with an offset of one byte, this code will be processed:

```
; CODE2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
CMP #$C9 ; 2
```

```
CMP #$C5 ; 2
NOP      ; 2
```

This makes 14 clock cycles, and only the status register will be changed. If the code is called with an offset of two bytes, it is started at the CODE1 segment at the second instruction. Add another one, you'll get to the second instruction of the CODE2 segment, and so on. This way it is possible to specify the exact number of clock cycles to be "wasted". With on exception: to be more specific there are 2 + X clock cycles that are wasted. There is no way to waste exactly one clock cycle.

Now we need a way to specify the "entry" of our "slide". On a C=64 this would be done using self-modifying code. The operand of a JMP $XXXX instruction will be replaced with the calculated address. This is not possible on systems like the Atari 2600, since the code is run in ROM. One option for example would be to use JMP ($0080) after writing the entry point to $0080 and $0081.

My approach differs a bit from the usual way. RAM is scarce on the Atari, and I don't want to "waste" up two of the 128 bytes available, when there is another way. When the CPU executes a JSR $XXXX (jump to subroutine) command, it writes the current address to the stack. To be more specific, it is the address of the JSR command + 2 which is the return address – 1. And this is what I do: I write my entry point – 1 to the stack and use the command RTS (return from subroutine) to jump into the clock slide. So, I'm still using two bytes of RAM, but only for a short time, without the need to evaluate which two bytes are available at this point.

```
; the X register specifies how many of the
; 15 clock cycles possible should be skipped
LDA #>clockslide
PHA
TXA
CLC
ADC #<clockslide
PHA
STA WSYNC ; <= this syncs to start of next scanline
clockslide:
RTS
CMP #$C9
CMP #$C9
CMP #$C9
CMP #$C9
CMP #$C9
CMP #$C9
CMP $EA
realcode:
; and here the real code continues
```

This approach still has one problem: between "clockslide" and "realcode", no page crossing may occur. If this were the case, I'd have to increase the high byte on the stack by one. But since the position of the code segments is under my control, I left this out as an exercise for the reader. ;-)

This entry was posted on Monday, February 6th, 2012 at 17:29 and is filed under 6502, tricks. You can follow any responses to this entry through the RSS 2.0 feed.

You can leave a response, or trackback from your own site.

2 Responses to “Clockslide: How to waste an exact number of clock cycles on the 6502”

George Phillips says:
7. February 2012 at 0:45

Very interesting. I had a similar cycle wasting problem on the Z-80. I tried various clockslide techniques but at a minimum of 4 cycles per instruction I couldn’t figure out how to make it work effectively. A series of $21 bytes might be a start (ld hl,$2121), but can’t see how it could be capped off.

Here’s my cycle waster for Z-80: http://members.shaw.ca/gp2000/beamhack3.html

I use it more for wasting out the rest of a frame so the high overhead is OK when hundreds or thousands of cycles are to be bled off.

DavidS says:
10. February 2012 at 12:21

WOW, way cool. I have been trying to get a simple clock slide for the C64 that does not waste a ton of code space and this is a perfect way to do so. Now with some miner modification I will be able to do much better (and faster) VIC II tricks..... Now the intro screen for my homebrew C64 clone will be amaizing (do you know how long it took to design a 100% VIC II compatable video chip with extended modes).

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Anti-spam word: (Required) *

To prove you're a person (not a spam script), type the security word shown in the picture.

Click on the picture to hear an audio file of the word.

Submit Comment