

# Flags on Decimal mode in the NMOS 6502

Version 1.0

Copyright © 2009 by Jorge Cwik.

<http://vapi.fxatari.com>

## Introduction

This article is about an undocumented behavior of the MOS 6502 CPU. Specifically, about the setting of the CPU flags after a decimal operation. More specifically, we'll describe the exact reason about the (apparently) very strange behavior of these flags after a decimal addition.

Official documentation states that after a decimal addition, only the C flag is valid. The state of the N, V, and Z flags are unpredictable or undocumented. However, long ago third party research found that the state of the flags is fully predictable.

According to most sources and current common knowledge, the behavior is as following. The Z flag is computed before performing any decimal adjust. In other words, the Z flag reflects the result of a binary addition. The N and V flags are computed after a decimal adjust of the low nibble, but before adjusting the high nibble.

The above behavior was verified by checking all the possible combinations. However, it is not really 100% accurate. You are probably asking, how a verified behavior could possibly be wrong? Well, what actually happens inside the processor is quite different, but the end result is the same and equivalent. So for the purpose of having a practical model to compute the flags, the known behavior is correct. For the purpose of understanding how and why things works at the hardware level, it is wrong. And that's what this article is about.

## Decimal mode

The 6502 has quite an unusual way to handle decimal mode. Most other processors either have separate instructions to perform decimal operations, or either use an extra specific instruction to adjust the result.

The 6502 uses the same instruction for both binary and decimal operations, and it doesn't need an extra instruction to adjust the result. This (almost) requires that the hardware would be able to perform the decimal adjust without any extra clock cycles.

Figure 1 is a simplified diagram of the 6502 ALU. The real ALU is more complex, we intentionally omitted most components that are not relevant to our purpose.

It might be surprising, but the diagram is enough to explain the odd flags behavior in decimal mode.

As was common in other 8-bit CPUs, the ALU is divided in two halves that are almost identical. Instead of one byte adder, we have two nibble adders. Each nibble adder computes a binary addition, a binary carry and a decimal carry. There is a mux at the output of each nibble adder that selects between the binary or decimal carry. The mux is controlled by the combination of the D flag and the current instruction. The outputs of each nibble adder go, optionally, through the decimal correction logic.

Note, in first place, that all the four ALU flags (C,N,V and Z) are computed at the same stage. This makes much more sense than computing some flags before decimal adjust, others in the middle, and some other ones after it. Note also that the computation of the decimal carry and the decimal adjust of the result are performed separately, and at different stages.

So what we see is that the hardware computes a binary addition, with both the half and byte carries being decimal. All the flags are computed from that result. This is different than you might assume considering the known behavior of the flags.

## The behavior of the flags

Let's see how this diagram explains the behavior of the flags. The Carry flag is almost obvious. This decimal carry is computed by dedicated logic, independently of both the binary adder and the decimal adjust.

The behavior of the N and V flags is not so obvious. They are supposed to be computed after decimal correction of the low nibble only, before adjusting the high nibble. But as we see in the diagram, no decimal correction has yet been done at this point.

The key is that the high nibble adder receives the decimal carry of the lower nibble. And obviously these flags (N and V) don't depend on the result of the lower nibble. They depend on the result of the high nibble only. In turn, the result of the high nibble is a simple binary addition. The low nibble contribution is indirect, and it is only by its output carry. The actual low nibble result, which is not yet adjusted, doesn't matter for the purpose of these flags. So even when no nibble was actually already adjusted at this point, these flags reflect the result as if the lower one (and only the lower one) was already adjusted.

Finally, let's consider the Z flag, which is supposed to be the same as in a binary addition. Again, this is not immediately apparent by the diagram. The Z flag is computed from the result of both nibble adders. The low nibble result will in effect always be the same regardless of the decimal mode. But the result of the higher nibble will not, it depends on the input carry being binary or decimal.

We can look and analyze this in several ways, including a verification of all the cases comparing the Z flag after a binary addition, with the actual addition performed by the hardware. One way for the analysis is described below.

The explanation comes from considering exactly when the output carry of the lower nibble (the half carry) is different in decimal than in binary mode. Remember that for all the cases where the half carry is the same in both modes, then the result of both nibble adders would also be the same in both modes.

The only cases when the half carry depends on the mode, is when then the nibble binary result is between ten and fifteen inclusively (10 to 15). In this case the half carry would be set in decimal mode but cleared in binary mode. If the result is smaller then the half carry would always be clear, and if the result is higher than 15 it would always be set. However, when the result of the low nibble is between 10 and 15, then the Z flag would be cleared, regardless of the result at the high nibble.

So the Z flag is set according to the result of both nibble adders. In most cases this result would be the same in both modes. And when the result is different, the difference doesn't matter for this purpose, the Z flag would be clear anyway.

## Conclusion

The odd behavior is mainly a consequence of separate dedicated logic to compute the decimal carry before the actual decimal correction. This was probably required for making the decimal correction logic smaller and then faster. The carry computing logic runs in parallel to the nibble adders, it doesn't have to wait for the adders result. The decimal correction however, operates on the result of the adders, and then it was important to make it as fast as possible. If it weren't for this pre-calculation of the decimal half carry, then the combinational path to compute the adjusted decimal result would be too long. A combinational path too long reduces the maximum frequency of the device.

For this purpose, the ALU computes a binary addition but with the half carry being decimal. This operation yields the odd flags behavior.

Figure: 1

